Q. why are we making generic classes

-

```
class Node<T>{
T data;
Node next;
}

class LinkedList{
Node head;

adddAtFirst();

}
```

```
class Stack<T>{
int top=-1
T data[];

push()
pop()
peek()
}
```

```
void add(n1, n2){

}
```

10.20+20.30
+ -> String -> concatination
employee + employee

Employee

DataStructuctures -> Containers that are used to store the data or data processing (FIFO, FILO)

Array -> index
Stack ->
List ->
SET ->
Hashtable ->
Map ->

Seraching, Sorting

A,B,C,D

Words

Sentences

Paragraphs

iland
island

Generics

Java, Language Fundamentals

Simple Programs

Functunalities

Projects

type inference

Type Erasure

till java 1.4

```
class Box {
    private Object ref;

    public void setRef(Object ref) {
        this.ref = ref;
    }

    public Object getRef() {
        return ref;
    }
}

Box b1 = new Box();
b1.setRef(new Integer(10));
integer i1 =(Integer) b1.getRef();
```

Generic Interface

Q. What is an interface and why to use it?
- Set of protocols -> methods
- Have common method design in the unrelated types

```
interface Acceptable{
void accept(Scanner sc);
}
```

```
class Date{

}

class Employee{

}
```

1. Comparable
2. Comparator

till java 1.4

```
interface Comparable{
        int compareTo(Object o);
}
```

```
class Employee implements Comparable{
int id;
double salary;

// this->e1, obj-> e2
int compareTo(Object obj){
        Employee e = (Employee) obj; //CCE
        if(this.salary > e.salary)
                return 5 ;//(+ve value)
        else if(this.salary< e.salary)
                return -2; //(-ve value);
        return 0;
}
}

Employee e1 = new Employee(1,10000);
Employee e2 = new Employee(2,8000);
//e1>e2 (compare e1 with e2)
int res = e1.compareTo(e2);

Date d1 = new Date(1,1,2001);
int res2 = e1.compareTo(d1);
```

java 1.5 onwards

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

```
class Employee implements Comparable<Employee>{
int id;
double salary;

// this->e1, obj-> e2
int compareTo(Employee obj){
        if(this.salary > e.salary)
                return 5 ;//(+ve value)
        else if(this.salary< e.salary)
                return -2; //(-ve value);
        return 0;
}
}

Employee e1 = new Employee(1,10000);
Employee e2 = new Employee(2,8000);
//e1>e2 (compare e1 with e2)
int res = e1.compareTo(e2);

Date d1 = new Date(1,1,2001);
int res2 = e1.compareTo(d1);// Compilation error
//Type Safety
```

interface Comparator

compares two given objects

```
interface Comparator{
int compare(Object o1, Object o2);
}

Employee e1 = new Employee(1,10000);
Employee e2 = new Employee(2,8000);

class EmpComparator implements Comparator{
      int compare(Object o1, Object o2){
      Employee e1 = (Employee) o1;
      Employee e2 = (Employee) o2;
      if(e1.salary > e2.salary)
            return 5 ;//(+ve value)
      else if(e1.salary< e2.salary)
            return -2; //(-ve value);
      return 0;
}
}
EmployeeComparator c = new EmployeeComparator();
c.compare(e1,e2);

   void sort(Object[] arr){
         if(arr[0] instanceof Comparable){
         for(int i=0;i<5;i++){

         if(arr[i].compareTo(arr[i+1])>0)
               swap(arr[i],arr[i+1])
   }
   }
   else
         throw new ClassCastException();
}
```

```
      interface Comparator<T>{
      int compare(T o1, To2);
      }

      Employee e1 = new Employee(1,10000);
      Employee e2 = new Employee(2,8000);

class EmpComparator implements Comparator<Employee>
      int compare(Employee o1,Employee o2){
      if(o1.salary > o2.salary)
            return 5 ;//(+ve value)
      else if(o1.salary< o2.salary)
            return -2; //(-ve value);
      return 0;
}
}

EmployeeComparator c = new EmployeeComparator();
c.compare(e1,e2);
```

Comparable<>

```
void sort(Object[] arr,Comparator<Employee> c){
      if(arr[0] instanceof Comparable){
      for(int i=0;i<5;i++){

      if(c.compare(arr[i],arr[i+1])>0)
            swap(arr[i],arr[i+1])
}
}
else
      throw new ClassCastException();
}
```

```
Before Sorting ->
Employee [empid=5, name=Mukesh, salary=40000.0]
Employee [empid=3, name=Suresh, salary=10000.0]
Employee [empid=4, name=Sham, salary=20000.0]
Employee [empid=1, name=Ramesh, salary=50000.0]
Employee [empid=2, name=Anil, salary=30000.0]
After Sorting on empid ->
Employee [empid=1, name=Ramesh, salary=50000.0]
Employee [empid=2, name=Anil, salary=30000.0]
Employee [empid=3, name=Suresh, salary=10000.0]
Employee [empid=4, name=Sham, salary=20000.0]
Employee [empid=5, name=Mukesh, salary=40000.0]
After Sorting on name ->
Employee [empid=2, name=Anil, salary=30000.0]
Employee [empid=5, name=Mukesh, salary=40000.0]
Employee [empid=1, name=Ramesh, salary=50000.0]
Employee [empid=4, name=Sham, salary=20000.0]
Employee [empid=3, name=Suresh, salary=10000.0]
```

Comparable
compareTo()

Comparator
compare()|

If we want to sort the array in natural ordering of elements use comparable interface

If we want to sort the array other than the natural ordering then use the comparator interface

Employee e1 = new Manager()
Employee e1 = new Salesman()

Manager m1 = new Employee(); // NOT OK

Q. Create a student class with name,rollno,marks
create a menu driven code that accepts the student and stores it inside the array.
display all the students.
display all students sorted on rollno in asc order
display all students sorted on name in asc order
display all students sorted on marks in desc order

Why to do overriding
1.
2.
3.