String -> Class in java.lang
Immutable ->
String Contant/Literal Pool

- Mutable Strings
StringBuffer
    - ThreadSafe

StringBuilder
    - Not Thread Safe

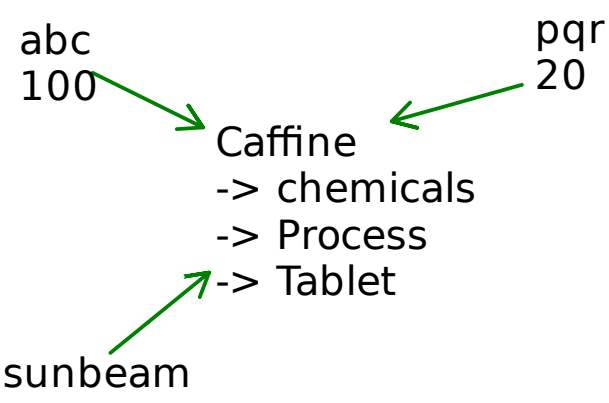
```
StringBuffer sb1 = new StringBuffer("sunbeam")
StringBuffer sb2 = new StringBuffer("sunbeam")
```

Enum ->
    Interally class
    PSF field =

```
enum ArithOps{
ADD,SUB,MUL,DIV
}
```

```
ArithOps.values()[0]
```

```
clone()
Employee e1 = new Employee();
Employee e2 = e1.clone(); // Shallow Copy
```



Generics

Logic -> Implemenation
By any type of Datatype

```
class Point{
xaxis;
yaxis;

}
```

```
swap(n1,n2){
temp = n1;
n1=n2;
n2=temp;
}
```

Genric interface

Generics
- If we want to write a logic that can work for any type of data type such code is called as generic code
- for eg -> swap()
- Generics in java is similar to template in cpp
- Generics was introduced in java from version 1.5
- before that generic implementation was done using Object class

1. using Object (1.4)

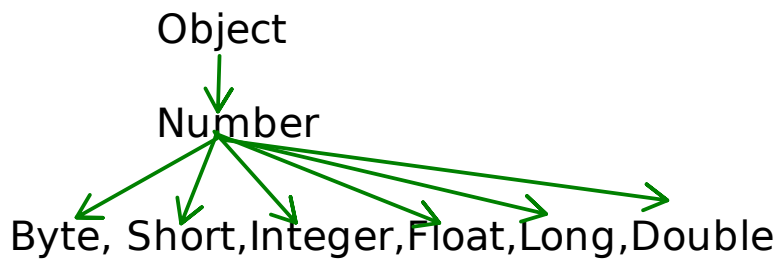
```
class Box{

Object obj;

void setObj(Object obj){
    this.obj=obj;
}

Object getObj(){
    return obj;
}
}

main(){
Box b1 = new Box();
b1.setObj(10);

Box b2 = new Box();
b2.setObj("Sunbeam");

sysout(value);
}
```

```
class Employee extends Object{

}

class Mobile extends Object{

}

class TV extends Object{

}
```

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

Object

```
Integer i1 = 10;
```



2. using generics (1.5)

```
class Box<T> {

T obj;

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

T getObj(){
        return obj;
}
}
```

1. Bounded Type Parameter
   -> Class types
2. Unbounded Type Parameter
   - Class References

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

Box b1

Box b2
```

```
class Object{


}

class Integer extends Object{

}
```

```
Integer i1 = 10;
Object obj = i1;
```

unbounded Types -> class Reference (?)

```
Box<? extends Number> b;
```

Number -> Upper Bound
- Byte
- Short
- Integer
- Float
- Long
- Double

```
Box<? super Integer>
```

- Number
- Integer <- Lower Bound

Generic Methods

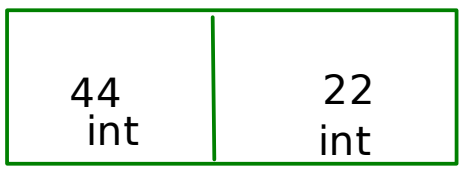
```
template<tyepname T>
void swap(T *ptr1,T *ptr2){
T temp = *ptr1;
*ptr1=*ptr2;
*ptr2=temp;

}

int num1=10;
int num2=20;
swap(&num1,&num2);
```

interface Box<T>

```
// using Object class
void swap(Object o1, Object o2){
Object temp = o1;
o1=o2;
o2=temp;
}

String o1 = "sun";
String o2 = "beam";
swap(o1,o2);

Employee e1 = new Employee();
Point p1 = new Point();
swap(e1,p1);
```

```
// using generics -> type Safe
void swap(Object o1, Objects o2){
T temp = o1;
o1=o2;
o2=temp;
}

String o1 = "sun";
String o2 = "beam";
<String>swap(o1,o2);

Employee e1 = new Employee();
Point p1 = new Point();
<Point>swap(e1,p1); // error
```

```
square(Box <Integer> num);
square(Box <String> num);
```



arr

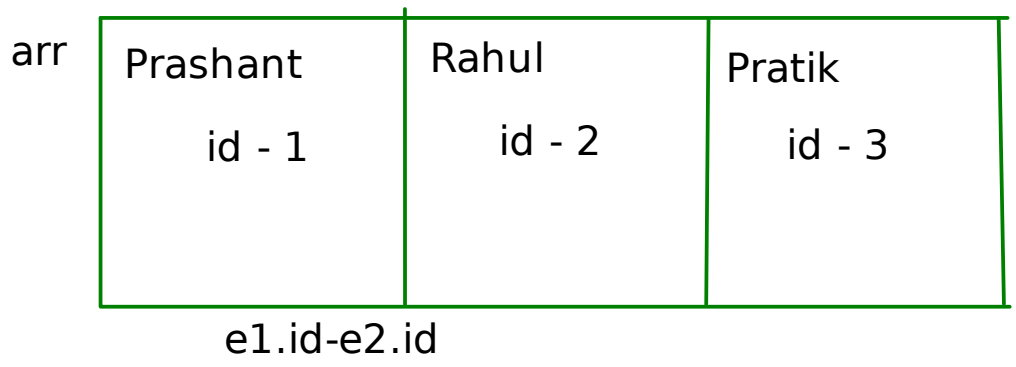
Ascending order

java.lang. Comparable

java.util.Comparator

```
Employee implements Comparable{
        int compareTo(T o){
// logic
}
}
```

```
Comaparable c1 = new Employee();
```




```
Comparable c1 = arr[i]; // Employee e1

Comparable c2 = arr[j]; // Employee e2;
```

```
 c1.comapreTo(c2)

  c1 == c2 -> 0
  c1>c2 -> +ve
  c1<c2 -> -ve
```

```
class Arrays{
static void sort(Object arr[]){
// selection sort
for(int i=0 ;i<3;i++){
        for(int j = i+1; j<3;j++)
        if(c1.compareTo(c2)>0){
        Employee temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
        }
}
}
```