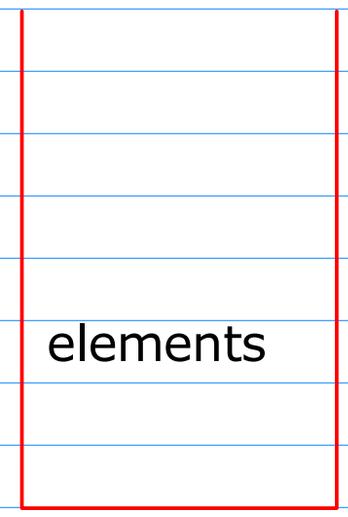
Generic / Template

- writing a code that can work for any type of data
- It is used to create the data structure classes

Stack,LinkedList

LIFO




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

void push(){
}

void pop(){
}

void peek(){

}

}
```

Logic

int,double,Employee, Manager,Student

till java 1.4

```
class Box {
        public Object obj; // any type of data inside it

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

        public Object getObj() {
                return obj;
        }
}
```

java 1.5 onwards

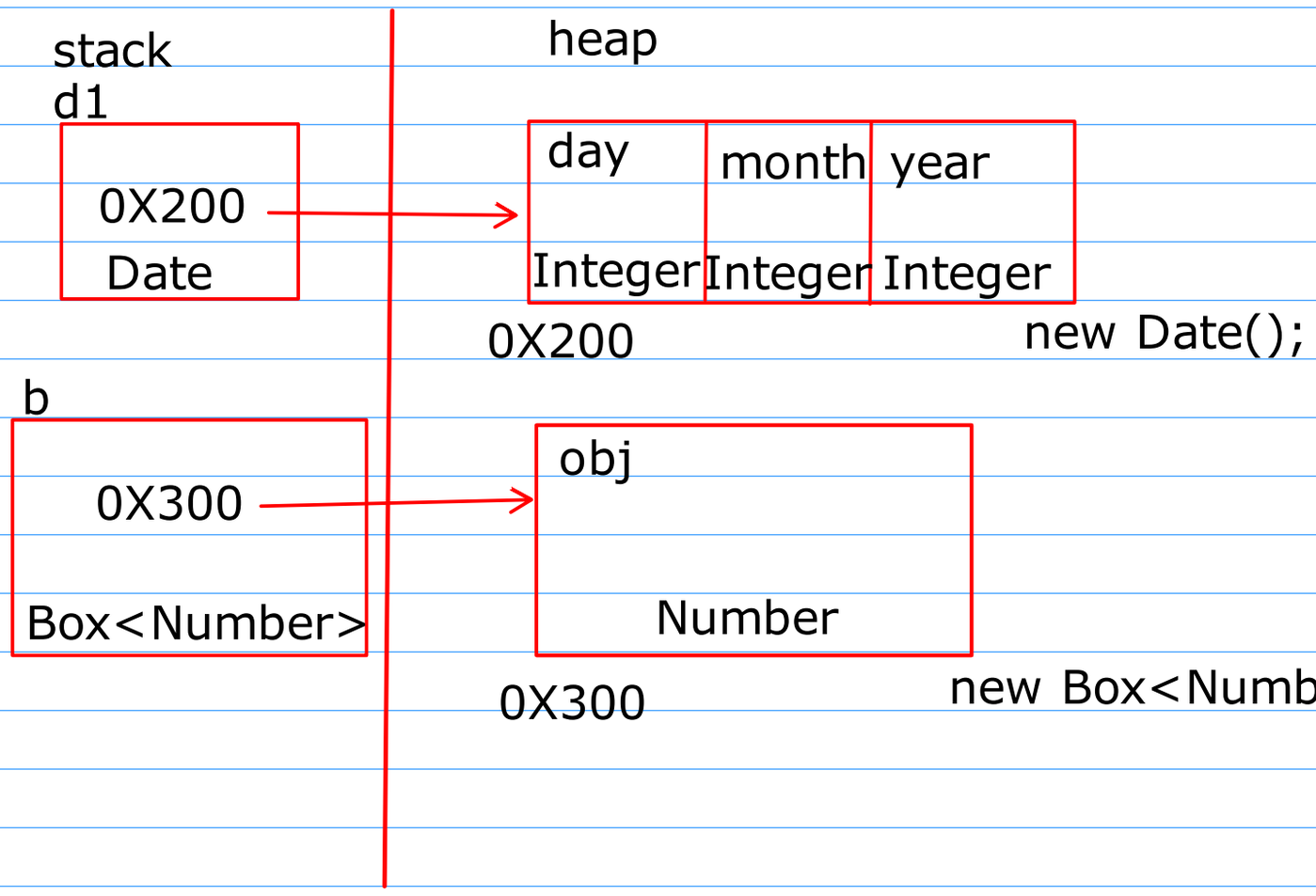
```
class Box<T> {
        public T obj; // any type of data inside it

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

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

```
// Generics
class Box<T> {
    public T obj; // any type of data inside it

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

    public T getObj() {
        return obj;
    }
}

public class Demo01 {
    public static void main(String[] args) {
        Box<Integer> b1 = new Box<Integer>();
    }
```

Integer

Type parameter

1. Bounded type parameter
   - It is used for classes
2. UnBounded type parameter
   - It is used for class references




```
class Box{
private Object obj;
}
```

```
class Box2{
private String obj;
}
```

```
class Object{

}

class String extends Object{

}
```

```
Box b1 = new Box2();
```

```
Stack<Manager> s1 =  new Stack<>();
s1.push(new Manage());
```

unBounded type parameter

- Is used only for class references

```
Box<Iphone> b1 = new Box<Iphone>();
Box<Redmi> b2 = new Box<Redmi>();
Box<Motorola> b3 = new Box<Motorola>();

Box<Keychain> b4 = new Box<Keychain>();
Box<Cups> b5 = new Box<Cups>();
```

```
class Box<T>{

}
```

```
// unbounded type parameter
distributer(Box<? extends Mobile> b){
// godown -> all boxes of Mobiles
}
```

```
class Employee{
id
name
salary
}
```

```
class Manager extends Employee{
bonus
}

class Salesman extends Employee{
noofproducts
commission
}
```

```
Box<Employee> b1 = new Box<Employee>()
LinkedList<Employee> l1 = new LinkedList<Employee>()
LinkedList<Manager> l2 = new LinkedList<Manager>()
LinkedList<Salesman> l3 = new LinkedList<Salesman>()
LinkedList<Integer> l4 = new LinkedList<Integer>()
```

```
void sort(LinkedList<? super Manager> l1){
// logic on salesman sorting based on
// noofproducts and commission does not exists

id,name,salary -> Employee
bonus -> Manager

manager m;
m.bonus
}
```

```
l2.add(new Salesman())
```

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

```
e1.id
e1.name;
e1.salary;
```

```
Manager m = (manager)e1
m. bonus
```

```
Box<Employee> b1 = new Box<Employee>()
LinkedList<Employee> l1 = new LinkedList<Employee>()
LinkedList<Manager> l2 = new LinkedList<Manager>()
LinkedList<Salesman> l3 = new LinkedList<Salesman>()
LinkedList<Integer> l4 = new LinkedList<Integer>()
```

```
displayDetails(LinkedList<Employee> l1){
        sysout(id)
        sysout(name)
        sysout(salary)
if(e instance of manager)
        sysout(bonus)
else
        sysout(totalcomm)
}
```

```
l1.add(new Employee())
l1.add(new Manager())
l1.add(new Salesman())
```

```
displayDetails(LinkedList<? extends Employee> l1){
        sysout(id)
        sysout(name)
        sysout(salary)
if(e instance of manager)
        sysout(bonus)
else
        sysout(totalcomm)
}
```

```
l1.add(new Employee())
l1.add(new Manager())
l1.add(new Salesman())

l2.

l3.
```

```
displayDetails(LinkedList<? super Manager> l1){
        sysout(e.id)
        sysout(e.name)
        sysout(e.salary)
        Manager m = (Manager) e
        sysout(m.bonus)
}
```

```
//l1.add(new Employee())
l1.add(new Manager())
l1.add(new Salesman())

l2.
```

```
displayDetails(l2);
displayDetails(l3); // NOT OK
```

```
displayDetails(l1);
```

```
swap(obj1,obj2){
temp = obj1;
obj1 = obj2;
obj2=temp;
}
```