Replace Typecode with Subclass/State

(examples)

Replace Typecode by Subclass

```
class Employee{
 type: int;
 const int ENGG = 1;
 const int SALESMAN = 2;

 Employee (int i) {type = I;}

 operation() {
   if (type == ENGG) {
     S1
   } else if (type==SALESMAN){
     S2
   }
 } }
Employee e = new Employee(1);
Employee s = new Employee (2)
```

→

```
class Employee {
 abstract operation();
}

class ENGG extends Employee {
   operation () { S1 }
}

class SALESMAN extends Employee {
   operation () {S2}
}

Employee e = new ENGG();
Employee s = new SALESMAN();
```

1. Instead of creating a common Employee object, create specialized ENGG / SALESMAN object
2. We avoid explicit conditional checks.
3. An ENGG remains an ENGG and does not become a SALESMAN or vice versa.
4. Instead of creating a vanilla Employee object, we create an ENGG or SALESMAN object.

Replace Typecode by State

```
class SrEmployee{
  type: int;
  const int manager  = 1;
  const int techlead = 2;

  operation() {
    if (type == manager) {
     S1
    } else if (type==techlead){
      S2
    }
  }
}
```

→

```
class Employee {
  EmpState state;
}
class EmpState {
  abstract operation();
}
class manager extends EmpState{
   operation () { S1 }
}

class techlead extends EmpState{
   operation () {S2}
}
```

1. An SrEmployee can be in either state “manager” or “techlead”.
2. Instead of creating a common SrEmployee “operation”, create specialized states of SrEmployees.
3. We avoid explicit conditional checks.
4. Replace calls to “emp.operation()” → “emp.state.operation()”