Solution 5 - More Behavioural Design Patterns

1) **State Design Pattern**

```
interface State {
  void printState();
}

class Started implements State {
  public void printState()
     {System.out.println("Engine is Started");}
}

class Stopped implements State {
  public void printState()
     {System.out.println("Engine is Stopped");}
}

 class Engine {

    State started = new Stopped();

    void setState(State s) {started = s;}

    void printState()
        { started.printState();}
}
```

2) **Observer Design Pattern**

```
interface Observer {
  void update(State s);
}

 class Engine {

    State started        = new Stopped();
    Collection observers = new ArrayList();

    private void enotify() {
      for (Iterator I = observers.iterator(); I.hasNext();)
          ((Observer)I.next()).update(started);
    }

    public void attach(Observer o) {observers.add(o);}

    void setState(State s) {started = s; enotify();}

    void printState() { started.printState();}
}
```

3) It's a **Template Design Pattern**

4) **Chain of Responsibility Pattern**

```
interface Precondition {
    void setNext(Precondition p);
    boolean ok();

}

class Battery implements Precondition {
    private Precondition next = null;

    public void setNext(Precondition p) {next = p;}

    public boolean ok() {
       if (!battery()) return false;
       if (next!=null) return next.ok();
       return true;
    }

    abstract boolean battery();
}

class Starter {
   private Precondition p;

  public void setPrecondition(Precondition p) {this.p = p;}

   public void start() {
      if (!p.ok()) return;
          (Engine.getEngine()).start();
   }

}
```