**GRASP**

GRASP is set of exactly 9 **G**eneral **R**esponsibility **A**ssignment **S**oftware **P**atterns.

GRASP Patterns help one understand essential object design & apply design reasoning in a methodical, rational, explainable way.

This is the list of 9 *GRASP* patterns (sometimes called principles)

1. Information Expert  
2. Creator  
3. Controller  
4. Low Coupling  
5. High Cohesion  
6. Indirection  
7. Polymorphism  
8. Pure Fabrication  
9. Protected Variations

### **1. Information Expert**

In following example Customer class has references to all customer Events so it is pivot candidate

to take responsibility of calculating total number of Events they have booked.

Event Creations are done if customer is present & without them events have no data. So Customer Class should be given responsibility to be an Information Expert.

public ArrayList<Event> getListOfEvents(String CustomerId) {

EventDB obj = new EventDB();

return obj.getListOfEvents();

}

}

Chart, box and whisker chart

Description automatically generated

2**. Creator**

**Customer is the Creator here as he is responsible for creating orders**

The Creator principle gives guidelines as to which class B should be in charge of creating a certain type of objects A. The principle contains a set of rules, such as:

B contains or aggregates instances of A

B closely uses A

B has the inputs to construct A and so on

The more rules are fulfilled, the better B is suited to instantiate objects of type A.

public void bookEvent(String cust\_id, String venue\_id, String studio\_id, String menu\_id, String catering\_id, String media\_id)

{

EventDB eventDB = new EventDB();

bill.setMenu\_cost(eventDB.getChosenMenuCost(menu\_id));

bill.setVenue\_cost(eventDB.getChosenVenueCost(venue\_id));

bill.setCatering\_cost(eventDB.getChosenCateringServiceCost(catering\_id));

bill.setStudio\_cost(eventDB.getChosenStudioCost(studio\_id));

price = this.getTotalBill();

eventDB.addEvent(name, type, date, total\_guests, price, starting\_time, ending\_time, cust\_id, venue\_id, studio\_id, menu\_id, catering\_id, media\_id, 0);

}

Diagram, box and whisker chart

Description automatically generated

**3.High Cohesion**

The principle of High cohesion encourages to focus classes around one responsibility, and to have all its components oriented towards achieving this responsibility. This is the principle of “do one thing and do it well”.

In the current scenario manager class is responsible for generating bill for respective event as it includes 3 separate costs i.e. electricity cost, security cost, staff cost where all three of them are set by manager .

Diagram

Description automatically generated

4)**Low coupling**

Coupling happens between two parts of the code when one depends on the other. Coupling introduces complexity, if only because the code can then no longer be understood isolation.

Such dependencies can be explicit, in terms of a function call for example, and that’s unavoidable and often OK.

Here Event is meant for Low Coupling as Event is independent of other objects & classes as it needs input from Manager but it has isolated functionality which leads to low coupling

Diagram

Description automatically generated