GRASP: an acronym (an abbreviation that can be pronounced) stands for General Responsibility Assignment Software Patterns.

Usage: The patterns are a general guideline for the designer to improve domain model understanding. Domain model contains candidate entities that will be converted into actual code. Thus, building a better understanding of domain model helps us in writing better and safer code.

Details: There are five design patterns in total in general GRASP. These are used to enhance understanding of how a candidate entity may transform in code. In order to better understand GRAPS think of it as a classification mechanism that puts each entity into a particular class. This classification tells us how to code a concept when the coding phase starts. The details of these guidelines are as follows.

A. **Creator**: "an entity responsible for creation of another is called creator entity. The entity created is called created entity."

Normally in a flow of functions some classes are responsible of calling constructor of other classes. These calling classes may or may not set parameters of the created instance. In either case the caller class has the code to call the constructor thus, is called the creator.

Create a table with columns labeled "creator" and "created entity". Write down the names of both inside the table with reasoning if any.

Example:

#	Creator	Created Entity	Reasoning
1	Customer	Order	Customer creates the instance of order.
2	Customer	Payment	Customer creates the instance of payment
3	Order Manager	Receipt	Order manager creates receipt for customer

B. **Controller**: "an entity responsible for manipulating execution flow of another entity"

Normally there are classes that decide what another class would do or behave by calling functions of that class or setting data inside of it. These classes are responsible for the life cycle of other classes.

Create a table with columns labeled "controller" and "controlled entity". Write down the names of both inside the table with reasoning if any.

Example:

#	Controller	Controlled Entity	Reasoning
1	Customer	Cart	Customer adds or removes products from cart
2	Customer	Payment	Customer makes payment exactly or less or more then the receipt value

C. Information Expert: “an entity holding information about/on another entity”

In a collection of classes working together as some classes control other classes similarly some hold information about other classes. This information can be used to set parameter or initiate other instances of classes. This information can be used to control flow of execution as well of other classes in case of a condition. Imagine a class that contains all the rules for a policy on resource allocation for new contacts. This class will be consulted whenever a new contact is created to interact with the server. Another good example is, the roadmap of students courses offerings. Each time a new semester starts this roadmap is consulted for course offerings. This way roadmap holds information on the semester courses.

Example:
Create a table with columns labeled “expert” and “information”. Write down the names of both inside the table with reasoning if any.

#	Expert	Information	Reasoning
1	Invoice	Order	What is added inside a order is recorded in a receipt
2	Receipt	Payment	Receipt holds information about the payments made

D. Cohesion: “the ability of a class to achieve its goal without over-depending (calling functions of another class over the limit(threshold)) on another class”

In a software almost all classes relate to each other in one or the other way. These connections are based on classes sharing data or behavior by calling each other functions. A threshold is defined to identify the count of these calls as over/under the allowed limit. Cohesion is defined if the number of calls to outside the class resources are under the limit.

E. Coupling: “the dependency of a class on another class(es) in order to achieve its objective by calling functions or sharing data defines its degree of coupling”

A class calls functions defined inside another class to get either data or behavior-based returns. These returns are used by the calling class to obtain results. In a collection of inter-related classes each one is coupled with one or more classes and yet is cohesive as well. Our concern as designers is to minimize coupling and maximization of cohesion.