CREATOR

Creator is a GRASP Pattern which helps to decide which class should be responsible for creating a new instance of a class. Object creation is an important process, and it is useful to have a principle in deciding who should create an instance of a class.

There are following 6 types of creational design patterns.

- ✓ Factory Method Pattern.
- ✓ Abstract Factory Pattern.
- ✓ Singleton Pattern.
- ✓ Prototype Pattern.
- ✓ Builder Pattern.
- ✓ Object Pool Pattern.

One of the most common activities in object oriented system is creation of objects. General principle is applied for the assignment of creation responsibilities.

Design supports :
1) low coupling
2) increased clarity
3) encapsulation
4) reusability.

Solution

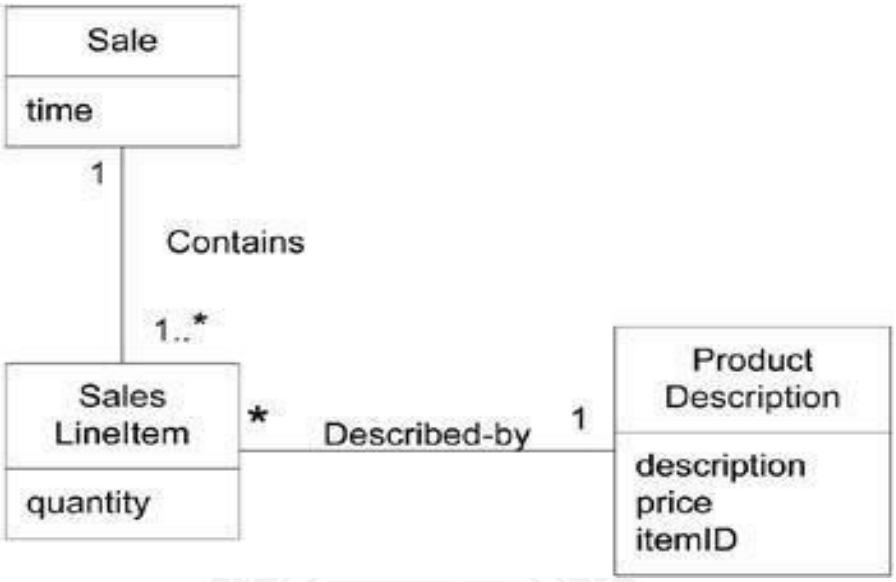
Assign class B the responsibility to create an instance of class A if one of these is true

- B "contains" or compositely aggregates A.
- B records A.
- B closely uses A.
- B is an expert while creating A (B passes the initializing data for A that is passed to A when created.)

B is a creator of A objects. If more than one option applies, usually prefer a class B which
aggregates or contains class A.

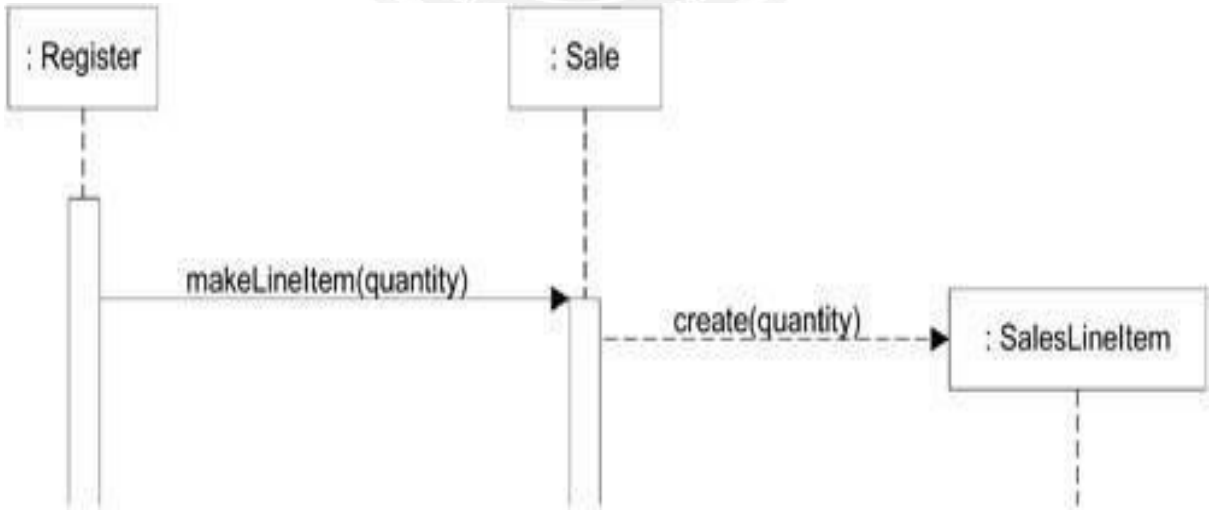
Example:

In the NextGen POS application, who should be responsible for creating a SalesLineItem instance? By Creator, we should look for a class that aggregates, contains, and so on, SalesLineItem instances. Consider the partial domain model



Partial Domain Model:

Here "Sale " takes the responsibility of creating 'SalesLineItem' instance . Since sale contains many 'SalesLineItem' objects.
The assignment of responsibilities requires that a 'makeLineitem' must also be defined in 'Sale'.



Creating a SalesLineItem:

Creator guides the assigning of responsibilities related to the creation of objects, a very common task. The basic intent of the Creator pattern is to find a creator that needs to be connected to the created object in any event. Choosing it as the creator supports low coupling.

All very common relationships between classes in a class diagram are,

- Composite aggregate part
- Container **contains** Content
- Recorder **records**

Creator suggests that the enclosing container or recorder class is a good candidate for the responsibility of creating the thing contained or recorded.

Example:- 'Payment' instance while creation initialized with 'sale ' total. 'Sale' is a candidate creator of 'Payment'.

Contradictions:

Based upon some external property value, creation requires significant complexity like,

- Recycled instances for performances.
- Creating an instance from one of a family of similar classes based upon some external property value, etc.

In such cases we go for helper class called

- **Concrete Factory, and**
- **Abstract Factory**

Benefits of the creator

- Low coupling is supported which implies Lower maintenance and higher opportunities for reuse

INFORMATION EXPERT (OR EXPERT)

Information expert (also expert or the expert principle) is a principle used to determine where to delegate responsibilities such as methods, computed fields, and so on. ... This will lead to placing the responsibility on the class with the most information required to fulfill it.

During Object Design, when the interactions between objects are defined, we make choices about the assignment of responsibilities to software classes. This makes the software easier to

- Maintain
- Understand
- Extend

Assign a responsibility to the information expert, the class that has the information necessary to fulfill the responsibility.

Example:-

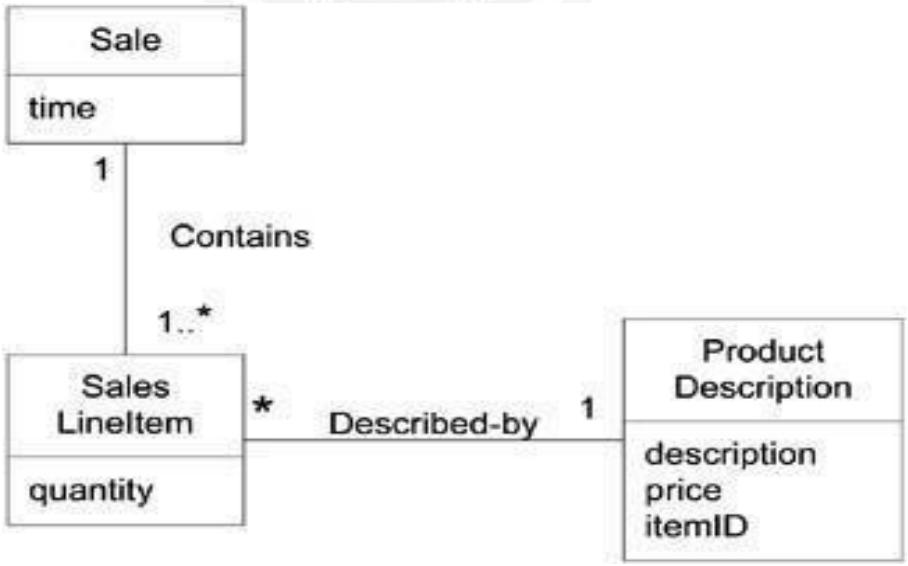
NextGEN POS Application

Some class needs to know the grand total of a sale. Start assigning responsibilities by clearly stating the responsibility.

1. Look at relevant classes in the Design Model if available,
2. Otherwise, look in the Domain Model

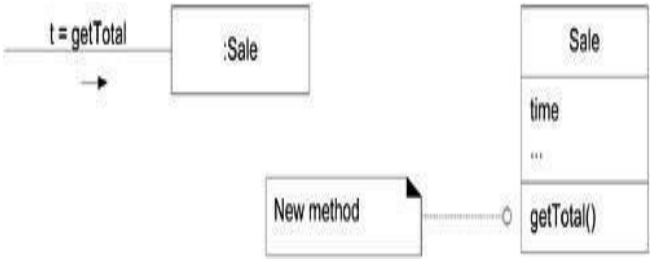
***Example*:-**

If the design work has been just started, then look into the domain model, the real-world "sale". In design model, software class 'sale' is added with the responsibility for getting total with the method 'getTotal'.



Partial domain model for association of sale

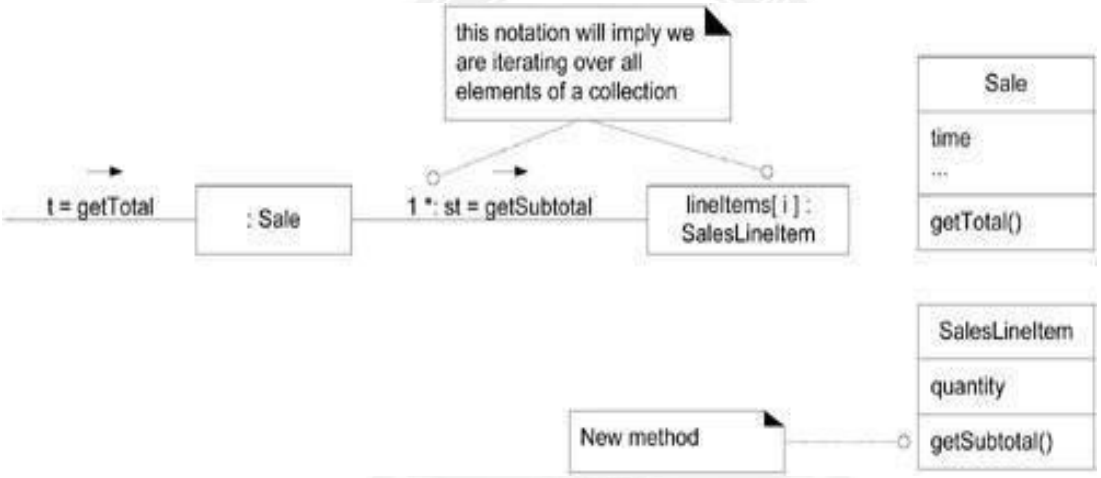
After adding the getTotal() , the partial interaction and class diagrams given as :



To determine expert using the above the SalesLineItem should determine subtotal.

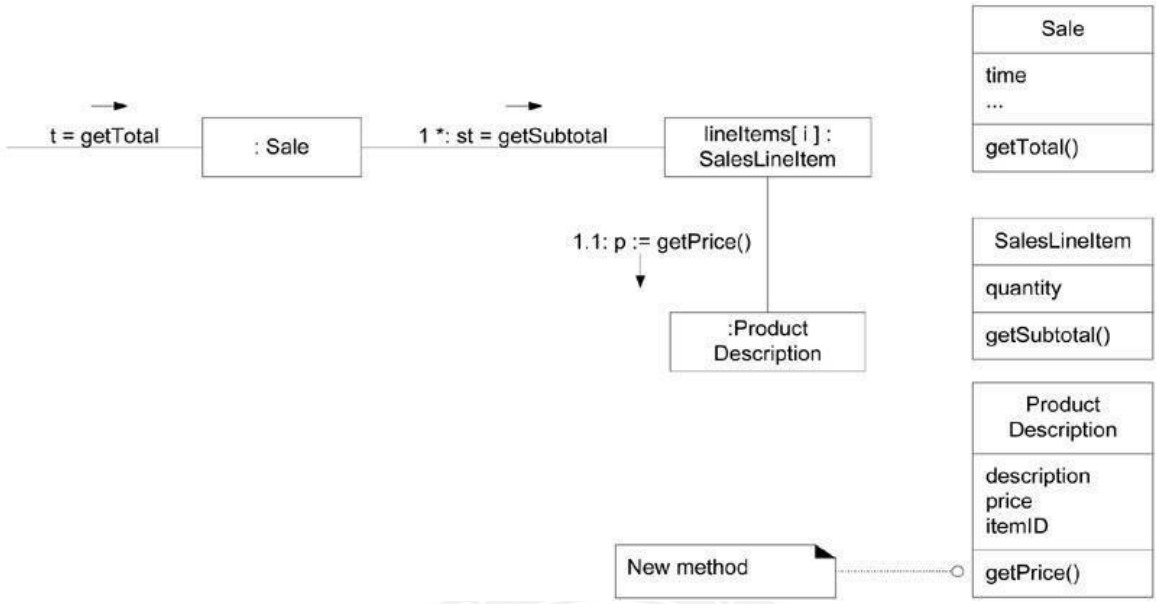
- SalesLineItem Quantity
- ProductDescription Price

By information expert using the above the Sales LineItem should determine subtotal. This is done by Sale sending get Subtotal messages to each Sales LineItem and sum the results.



Calculating Sales Total

After knowing and answering subtotal , a SalesLineItem sends it a message asking for the product price. ProductDescription is an information expert on answering its price.



Calculating the sale total

Finally, we assigned three design classes of the objects with three responsibilities to find the sales total.

Design Class	Responsibility
Sale	knows sale total
SalesLineItem	knows line item subtotal

Design Class	Responsibility
ProductDescription	knows product price

- The Information Expert is frequently used in the assignment of responsibilities
- Experts express the common "intuition" that objects do things related to the information they have.
- Partial information experts will collaborate in the task.
- For example:- Sales total problem experts will collaborates in the task.
- Information expert thus has real world analogy.
- Information experts are basic guiding principle used continuously in object design.

Contradictions

Solution suggested by Expert is undesirable, usually because of problems in coupling and cohesion.

To overcome this,

- Keep application logic in one place(like domain software objects)
- Keep database objects in another place(separate persistence services subsystem.
- Supporting a separation of major concerns improves coupling and cohesion in a design.

Benefits

• Information encapsulation is maintained since objects use their own information to fulfill tasks.

• High cohesion is usually supported