

Design Patterns

An Introduction



Motivation

- Expert designers do not solve every problem from first principles.
- Solutions are reusable.
- There are recurring patterns of classes and communicating objects that solve specific design problems and make these designs more flexible and reusable.



What is a Design Pattern?



- A design pattern systematically names, explains and evaluates an important and recurring design in object-oriented systems.
- It is possible to collect these patterns into a catalog for designers to use to select and evaluate alternatives.

Essential Elements of a Design Pattern



Pattern Name



- A handle to describe a design problem, its solutions and consequences.
- Allows for design at a higher level of abstraction.
- Communication mechanism for designers.

Problem



- Describes when to apply the pattern.
- Explains the problem and its context.
- May include a list of conditions that must be met before it makes sense to apply the pattern.



Solution

- Describes the elements that make up the design, their relationships, responsibilities and collaborations.
- It is not the design or implementation.
- A pattern is like a template.
- Provides an abstract description of a design problem and how a general arrangement of elements solves it.



Consequences

- Results and trade-offs of applying the pattern.
- Critical for evaluating design alternatives and understanding costs and benefits of the pattern.
- The pattern's impact on the system's flexibility, extensibility or portability may be considered.

A More Detailed Look at Design Patterns



Definition



- Design patterns are not at the level of linked lists and hash tables nor are they application generators.
- They are descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context.

Definition



- A pattern identifies the classes and instances, their roles and collaborations and the distribution of responsibilities.
- Each one focuses on a particular problem or issue.
- A pattern may also provide sample code to illustrate an implementation.

Describing Design Patterns





Pattern Description

- **Pattern Name and Classification**
 - Important because it becomes part of your design vocabulary.
- **Intent**
 - What does the design pattern do?
 - What is its rationale and intent?
 - What particular design issue or problem does it address?



Pattern Description

- **Also Known As**
 - Other well-known names for the pattern, if any.
- **Motivation**
 - A scenario that illustrates a design problem and how the class and object structures in the pattern solve the problem.



Pattern Description

- **Applicability**
 - When should the pattern be applied?
 - How can you recognize these situations?
- **Structure**
 - A graphical representation of the classes in the pattern using a notation such as UML to illustrate sequences of requests and collaborations.



Pattern Description

- **Participants**
 - The classes and/or objects participating in the design pattern and their responsibilities.
- **Collaborations**
 - How the participants collaborate to carry out their responsibilities.



Pattern Description

- **Consequences**
 - How does the pattern support its objectives?
 - What are the trade-offs and results of using the pattern?
 - What aspect of system structure does it let you vary independently?



Pattern Description

- **Implementation**
 - What pitfalls, hints, or techniques should you be aware of when implementing the pattern?
 - Are there language-specific issues?
- **Sample Code**
 - Code fragments that illustrate how you might implement the pattern in a particular object-oriented language.

Pattern Description



- **Known Uses**
 - Examples of the pattern found in real systems.
- **Related Patterns**
 - What design patterns are closely related to this one?
 - What are the important differences?
 - With which other patterns should this one be used?

How to Select a Design Pattern



Consider how design patterns solve design problems

Find appropriate objects.
Determine object granularity.
Specify object interfaces.



Scan Intent Sections



Study how patterns
interrelate



Study patterns of like
purpose

Study similarities and
differences.



Examine a cause of redesign

Consider what might force a change to your design.



Consider what should be variable in your design

Consider what you want to be able to change without redesign - encapsulate the concept that varies.



How to Use a Design Pattern



Read

- Read the pattern once through for an overview.
 - Pay special attention to the Applicability and Consequences sections.
 - Ensure the pattern is right for your problem.



Study



- Go back and study the Structure, Participants and Collaborations sections
- Look at the sample code.

Build



- Choose names for pattern participants that are meaningful in the application context.
- Define the classes.
- Define application-specific names for operations in the pattern.
- Implement the operations to carry out the responsibilities and collaborations in the pattern.

The GRASP Patterns



GRASP



- General Responsibility Assignment Software Patterns
 - Important to “grasp” these principles in order to design object-oriented software.

First Five Patterns

- Expert
- Creator
- High Cohesion
- Low Coupling
- Controller



Expert





Expert

- Problem
 - Who should be responsible for knowing some piece of information?
- Solution
 - Assign the responsibility to the information expert, i.e. the class that has the information necessary to fulfill the responsibility.



Benefits of Using Expert

- Maintains encapsulation, since objects use their own information to fulfill tasks.
- Supports low coupling, which leads to more robust and maintainable systems.
- More cohesive, “lightweight” class definitions.
 - Easier to understand.
- High cohesion

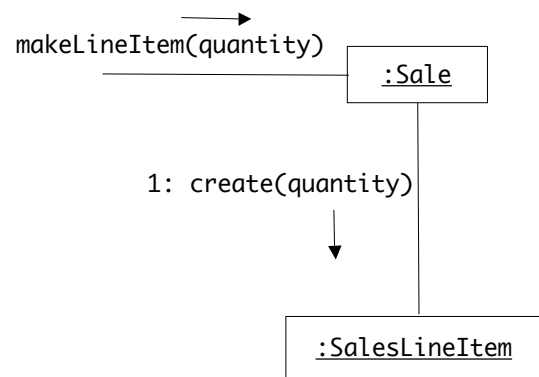


Expert – Also Known As

- Place responsibilities with data.
- That which knows, does.



Expert Example from the Text



Expert Example from the Text



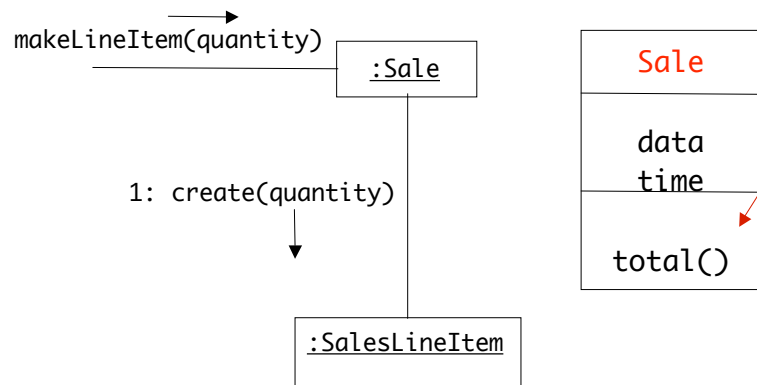
- Who should be responsible for knowing the grand total of a sale.
- What information is needed to determine the grand total?
 - Need to know all the **SalesLineItem** instances.
 - Need to know sum of subtotals of **SalesLineItem** instances.

Expert Example from the Text



- **Sales** is responsible for knowing the total.
- We may decide to incorporate a method within **Sale** to compute the total.
 - The Expert pattern lets you decide where to put your method.

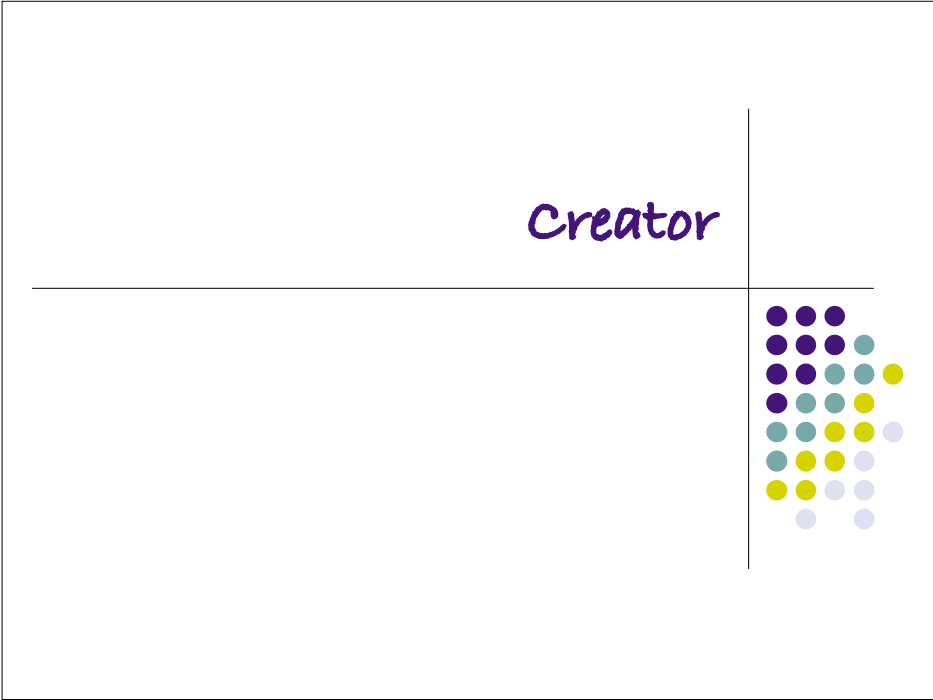
Expert Example from the Text



Expert Example from the Text



- What about line item sub-totals?
 - Who's the expert here?
- We need to know quantity and unit price to compute the subtotals.
 - **SalesLineItem** knows this!
 - Therefore, add a subtotal method to **SalesLineItem**.



Creator



- Problem
 - Who should be responsible for creating a new instance of a class?
- We need a general principle for the assignment of creation responsibility.



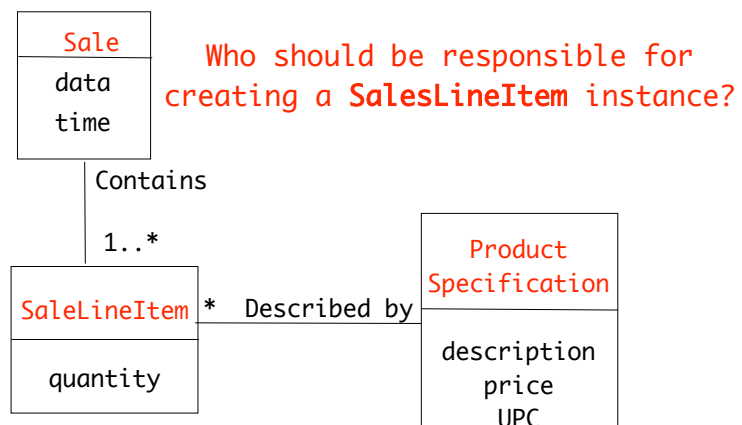
Creator

- Solution

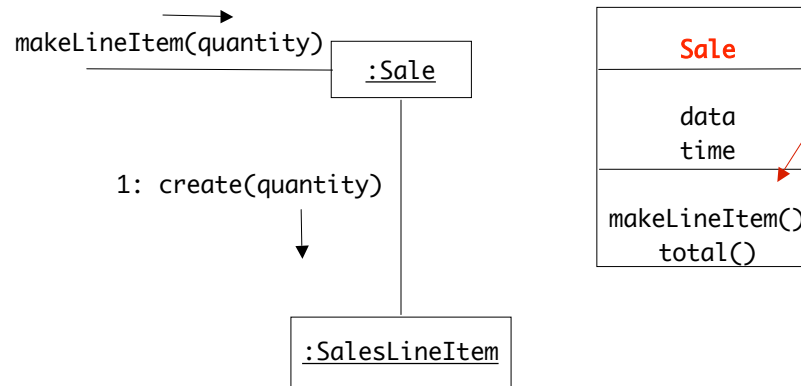
- Assign a class B the responsibility to create an instance of another class A, if one of the following is true:
 - B aggregates A objects
 - B contains A objects
 - B records instances of A objects
 - B closely uses A objects
 - B has the initializing data for A



Creator Example from Text



Creator Example from Text



Creator

- Supports
 - low coupling
 - increases clarity
 - encapsulation
 - reusability
- Low coupling – created class is already visible to the creator class.