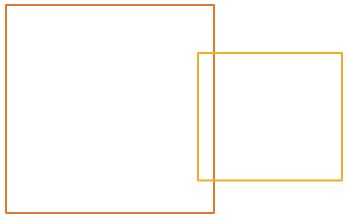
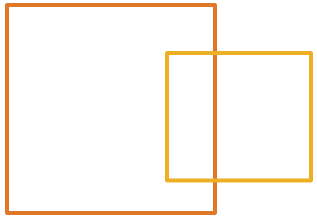
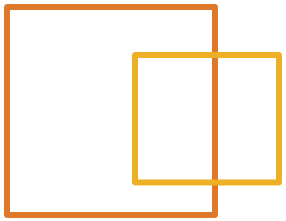
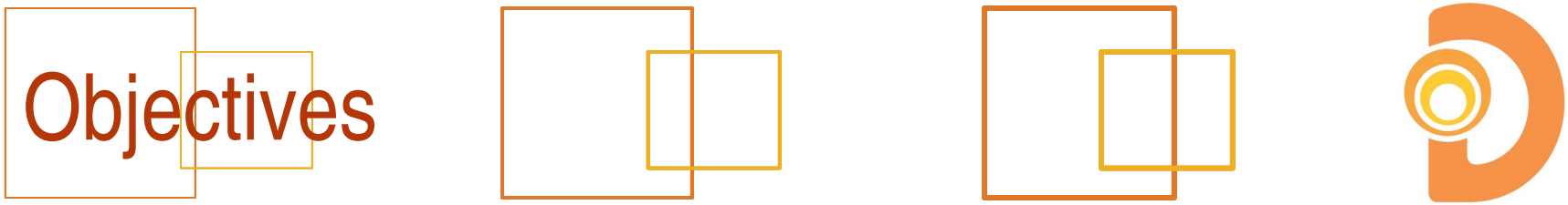




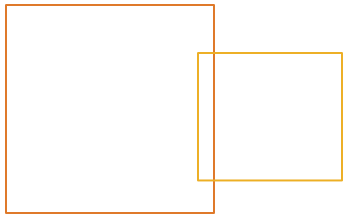
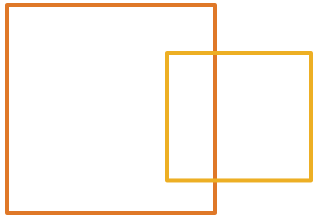
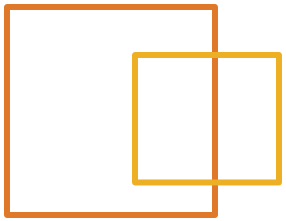

Object Oriented Design Patterns

© DevelopIntelligence http://www.DevelopIntelligence.com

Objectives



- On completion of this module you will be familiar with key patterns in the “Gang of Four” pattern catalog


Creational Patterns

What are Creational Patterns

Intent: Lower couplings and provide flexibility when it comes to creating objects

- **Motivation:**
  - Abstract the instantiation process
  - Hide how instances of these classes are created and assembled
  - Hide references to concrete classes used in the system
  - Govern the what, when, who, and how of object creation
- General example:
  - User Interface toolkit support for the Java Virtual Machine
  - AWT or Swing
  - Native-peer or 100% Java
  - Platform look and feel or custom look and feel

Factory Method Pattern Description

- **Intent**:
  - Define a way to create objects without knowing type
  - Defer instantiation to subclasses
  - **AKA**: Virtual Constructor
- **Motivation**:
- A database system needs to support multiple database drivers. The application can connect to different databases at runtime based on different driver. Because driver type is not known at compile time, application must find the right driver at runtime.

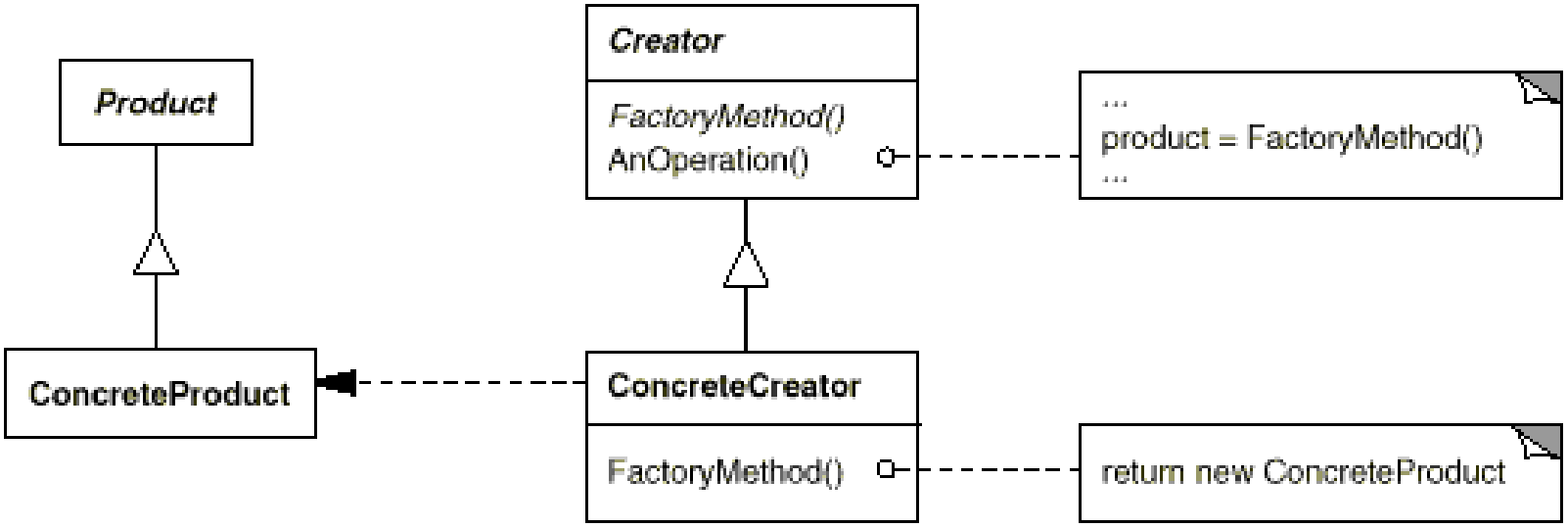
Factory Method Pattern Description

- Applicability:
  - Class can't anticipate the class of the object it must create
  - A class wants it subclass to specify the object it creates
  - Classes delegate responsibility to one of several helper subclasses, and you want to localize the knowledge of which helper subclass is the delegate.

Factory Method Software Example

- JDBC DriverManager.getConnection

Factory Method Pattern UML



Abstract Factory Pattern Description

- Intent: provide an interface for creating families of objects without specifying concrete classes
- AKA: Kit
- Motivation: System needs to load in a different UI toolkit at runtime based on system level variables
- Applicability:
  - Want to let system determine how to create family of objects
  - Want to let system determine which family of objects to create
  - Want to reveal only interface of a library

Abstract Factory Real World Example

This pattern is found in the sheet metal stamping equipment used in the manufacture of Japanese automobiles.

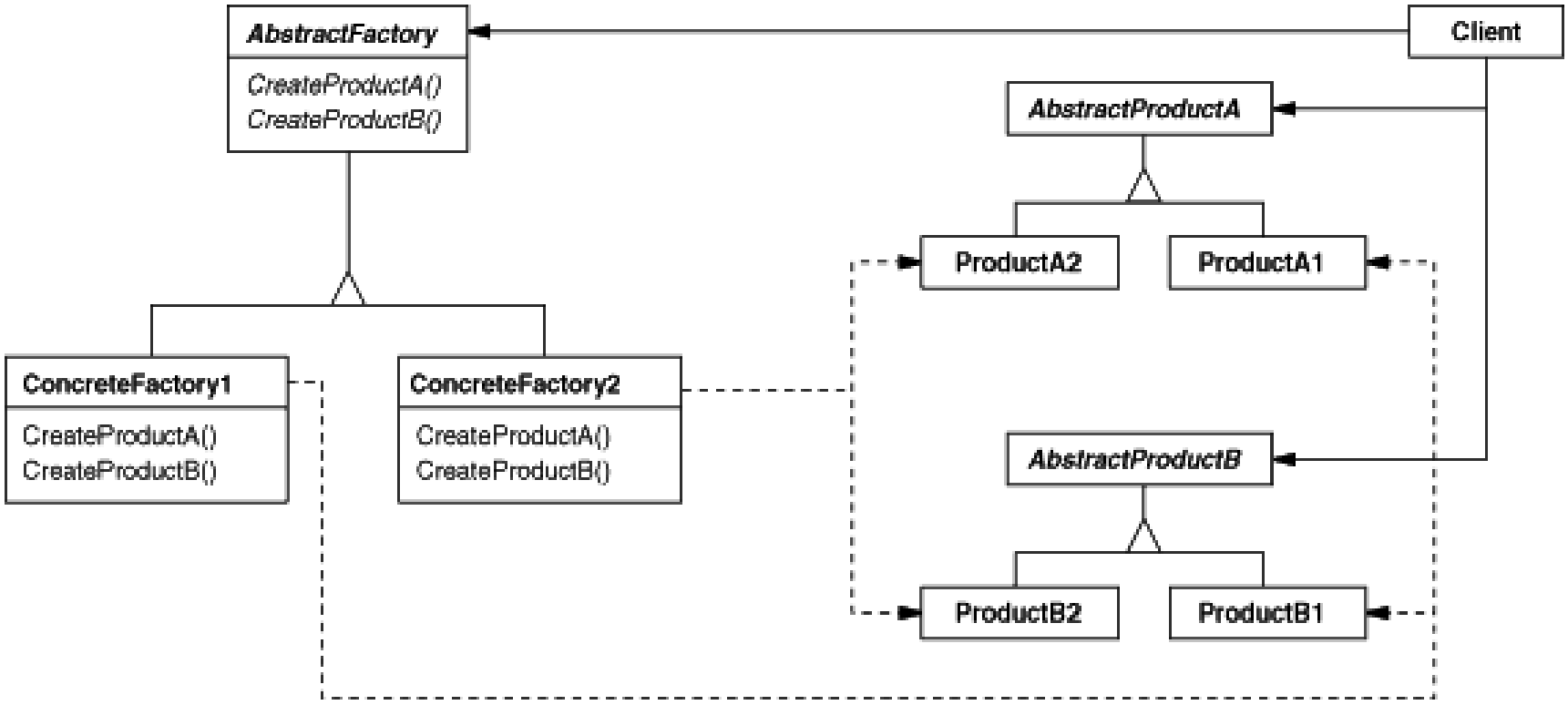
The stamping equipment is an *Abstract Factory which creates auto body parts. The same machinery is used to stamp right hand doors, left hand doors, right front fenders, left front fenders, hoods etc. for different models of cars.*

Through the use of rollers to change the stamping dies, the concrete classes produced by the machinery can be changed within three minutes

Abstract Factory Software Example

- JDBC DriverManager
- Java Swing PLAF

Abstract Factory Pattern UML



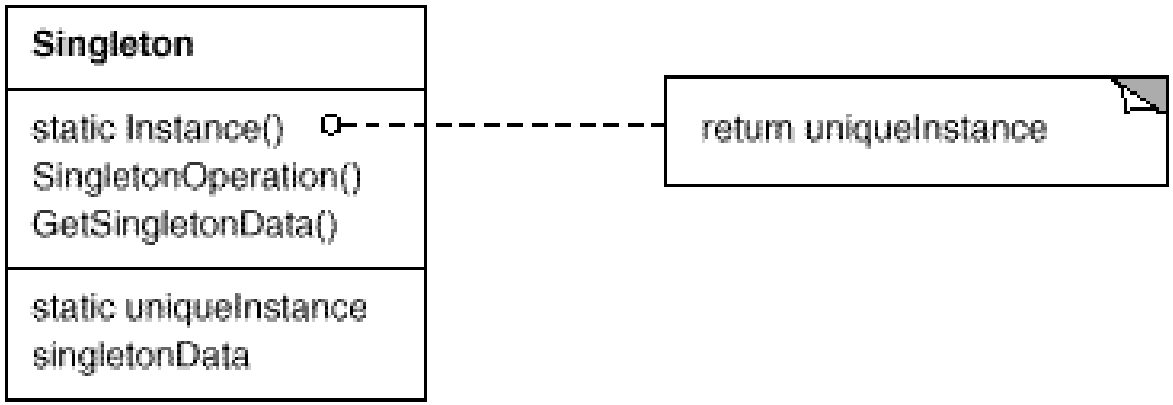
Singleton Pattern Description

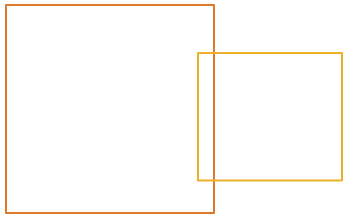
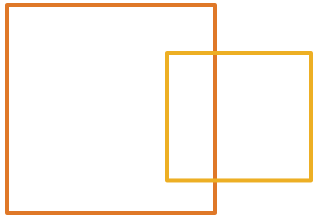
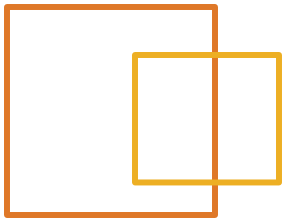
- Intent: Ensure a class only has one instance, and provide a global point of access to it.
- AKA: N/A
- Motivation: A database only supports one active connection.
- Applicability:
  - There must be exactly one instance of a class, and it must be accessible to clients from a well-known access point.
  - When the sole instance should be extensible by sub-classing, and clients should be able to use an extended instance without modifying their code

Singleton Software Example

- System.in, System.out, System.err
- AWT Thread

Singleton Pattern UML




Behavioral Patterns

What are Behavioral Patterns

- Describe algorithms, assignment of responsibility, and interactions between objects (behavioral relationships)
  - Behavioral class patterns use inheritance to distribute behavior
  - Behavioral object patterns use composition
- General example:
  - Model-view-controller in UI application
  - Iterating over a collection of objects
  - Comparable interface in Java

Observer Pattern Description

- Intent: Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically
- AKA: Dependents, Publish-Subscribe
- Motivation: An object-relational-mapping framework needs to keep track of changes made in a database and map them to an object and vice versa

Observer Pattern Description

- Applicability:
  - When a change to one object requires changing others, and you don't know how many objects need to be changed.
  - When an object should be able to notify other objects without making assumptions about who these objects are

Observer Real World Example

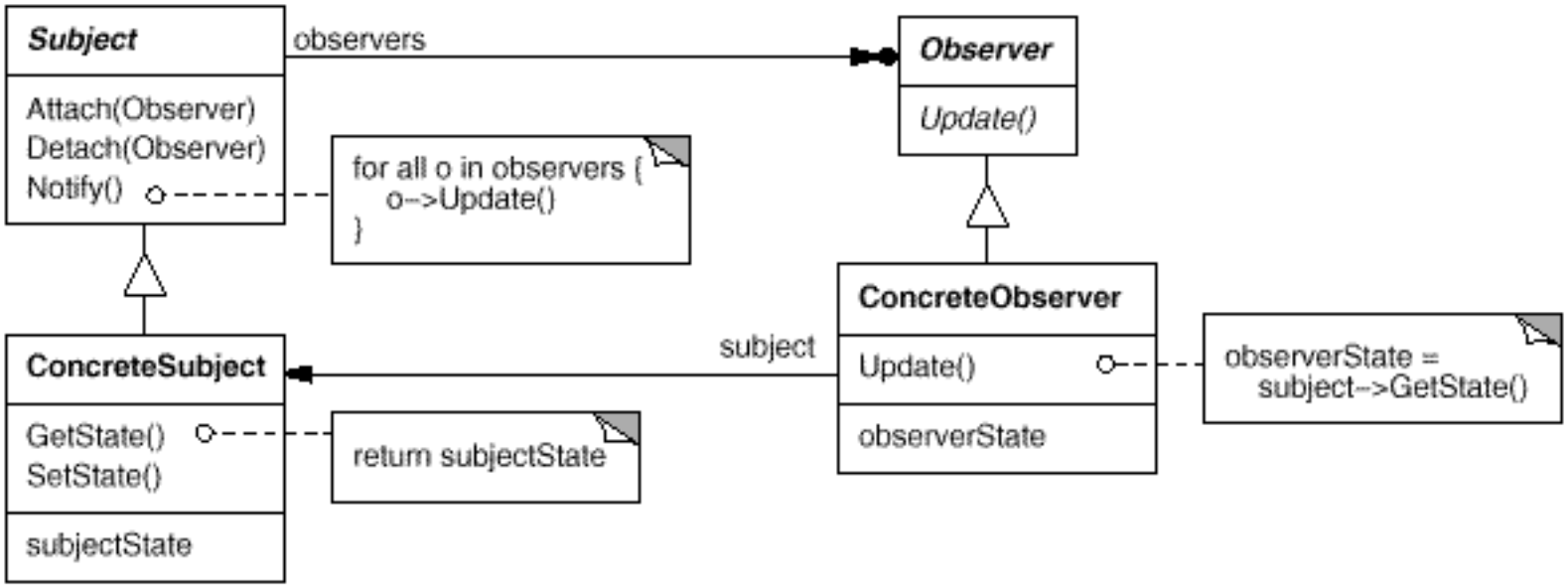
- The *Observer defines a one to many relationship, so that when one object changes state, the others are notified and updated automatically.*

- *Some auctions demonstrate this pattern. Each bidder possesses a numbered paddle that is used to indicate a bid. The auctioneer starts the bidding, and "observes" when a paddle is raised to accept the bid. The acceptance of the bid changes the bid price, which is broadcast to all of the bidders in the form of a new bid.*

Observer Software Example

- Most user interface toolkits
- Object-relational-mapping framworks

Observer Pattern UML



Strategy Pattern Description

- Intent: Define a family of algorithms, encapsulate each one, and make them interchangeable.
- AKA: Policy
- Motivation: A system needs to change how it connects to the web depending on the underlying security within an organization. It may be able to connect directly, use a proxy, or require security.
- Applicability:
  - You need different variants of an algorithm.

Strategy Real World Example

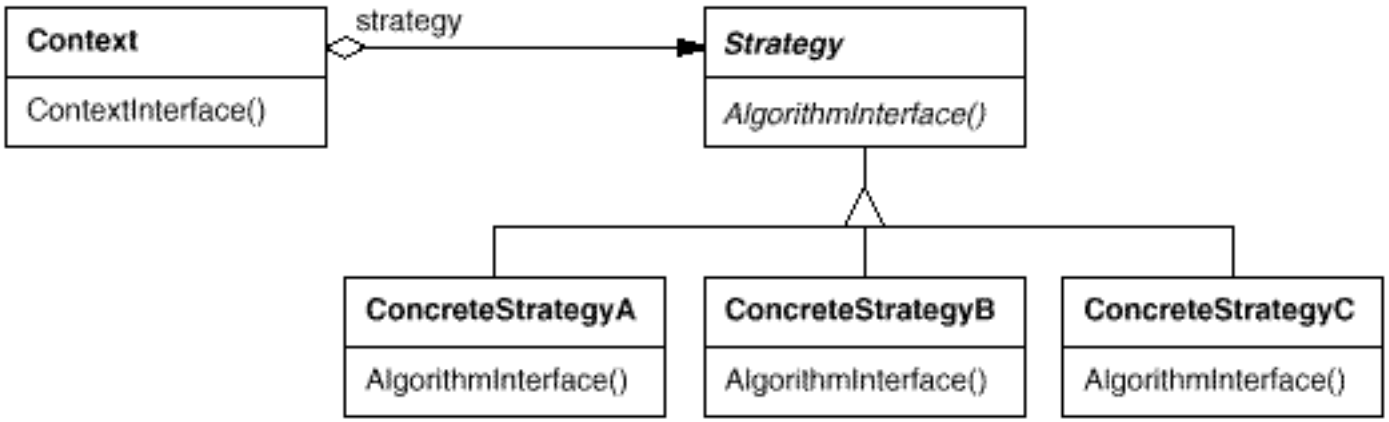
- A *Strategy defines a set of algorithms that can be used interchangeably.*

- *Modes of transportation to an airport is an example of a Strategy. Several options exist, such as driving one's own car, taking a taxi, an airport shuttle, a city bus, or a limousine service. For some airports, subways and helicopters are also available as a mode of transportation to the airport. Any of these modes of transportation will get a traveler to the airport, and they can be used interchangeably. The traveler must chose the Strategy based on tradeoffs between cost, convenience, and time.*

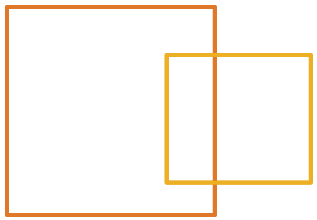
Strategy Software Example

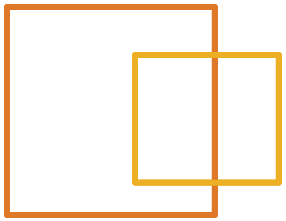
- Comparator / Comparable interfaces in Java collections

Strategy Pattern UML



Summary





- Now that we have completed this module you should be familiar with key patterns in the “Gang of Four” pattern catalog