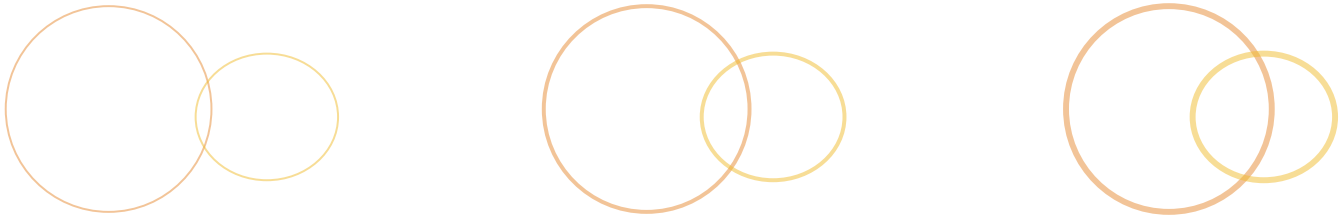


Creational Patterns

What are Creational Patterns

- Make the system independent of how objects are created, composed, and represented
- Abstract the instantiation process
  - Encapsulate the knowledge about concrete classes system uses
  - Hide how instances of these classes are created and assembled
- 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

List of Creational Patterns

- Class scope pattern:
  - Abstract Factory
- Object scope patterns:
  - Builder
  - Factory Method
  - Prototype
  - Singleton

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.

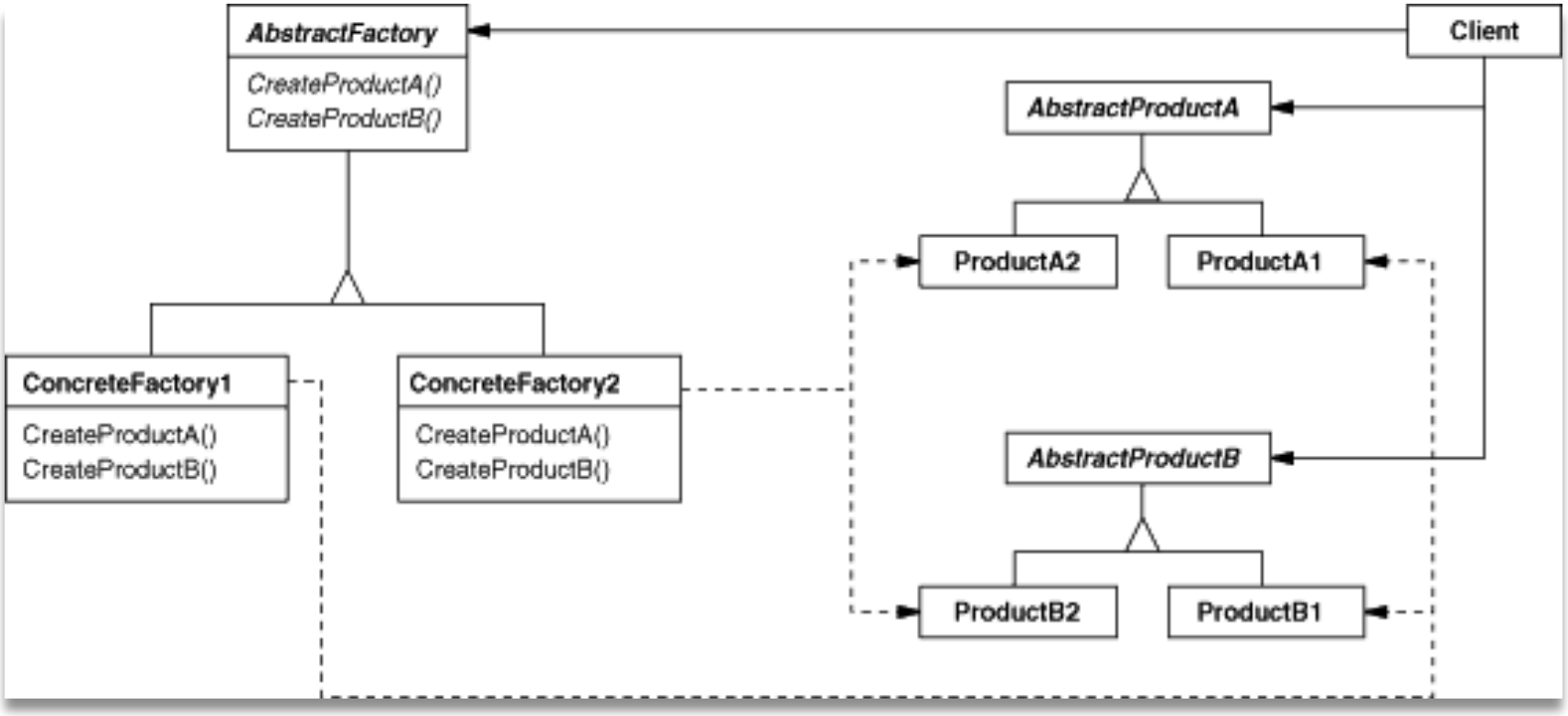
The stamping equipment is an *Abstract Factory which creates auto body parts. The same machinery is used to stamp right hand doors, left hand doors, right front fenders, left front fenders, hoods etc. for different models of cars.*

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



Builder Pattern Description

- Intent: separate the construction of a complex object from its representation so that the same construction process can create different representations.
- AKA: N/A
- Motivation: The system needs to read and write data to an underlying operating system. The problem is that each operating system handles I/O differently. Need a solution that allows the “handler” to be changed
- Applicability:
  - The algorithm for creating a complex object should be independent of the parts that make up the object and how they’re assembled
  - The construction process must allow different representations for the object that’s constructed

Builder Real World Example

The *Builder pattern separates the construction of a complex object from its representation, so that the same construction process can create different representation.*

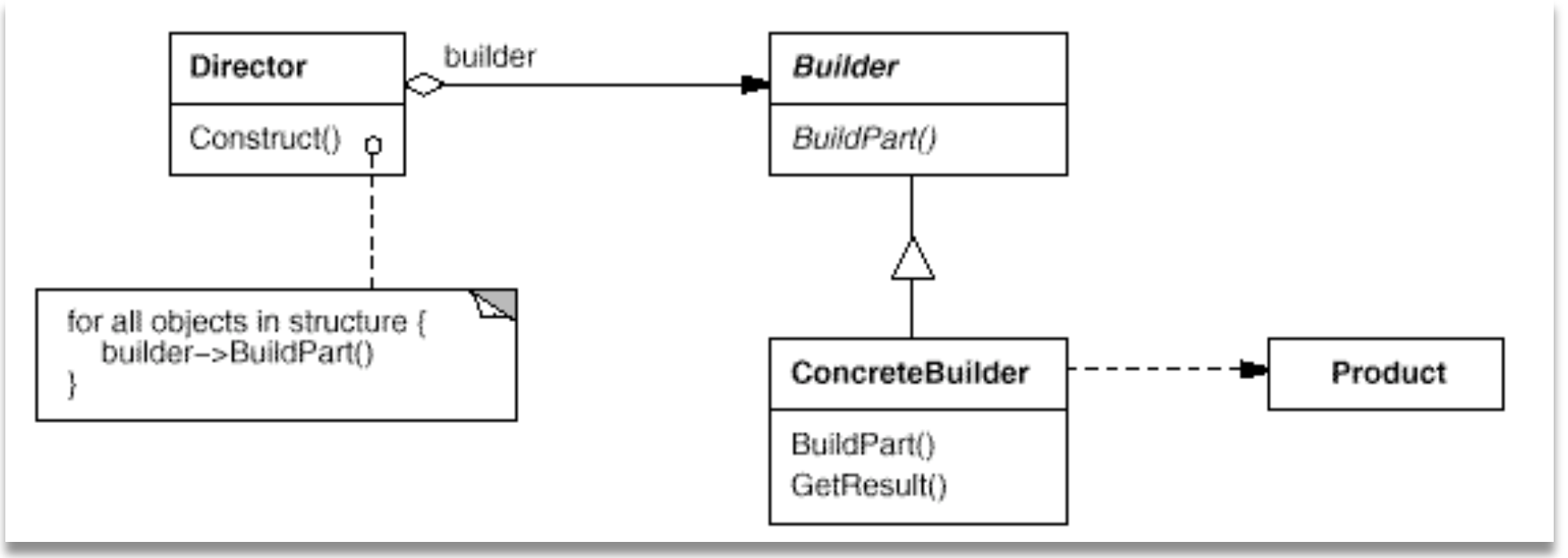
This pattern is used by fast food restaurants to construct children's meals. Children's meals typically consist of a main item, a side item, a drink, and a toy (e.g., a hamburger, fries, coke, and toy car). Note that there can be variation in the contents of the children's meal, but the construction process is the same.

Whether a customer orders a hamburger, cheeseburger, or chicken, the process is the same. The employee at the counter directs the crew to assemble a main item, side item, and toy. These items are then placed in a bag. The drink is placed in a cup and remains outside of the bag. This same process is used at competing restaurants.

Builder Software Example

- Java I/O implementation
- Creating an Input Stream delegates the creation of an underlying OS Specific input stream to native a library
- Construction process is exactly the same from client perspective
- How the object is created may differ

Builder Pattern UML



In UML a diamond represents aggregation

Factory Method Pattern Description

- Intent: define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class 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 and can anticipate which class will represent the connection.
- 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 Real World Example

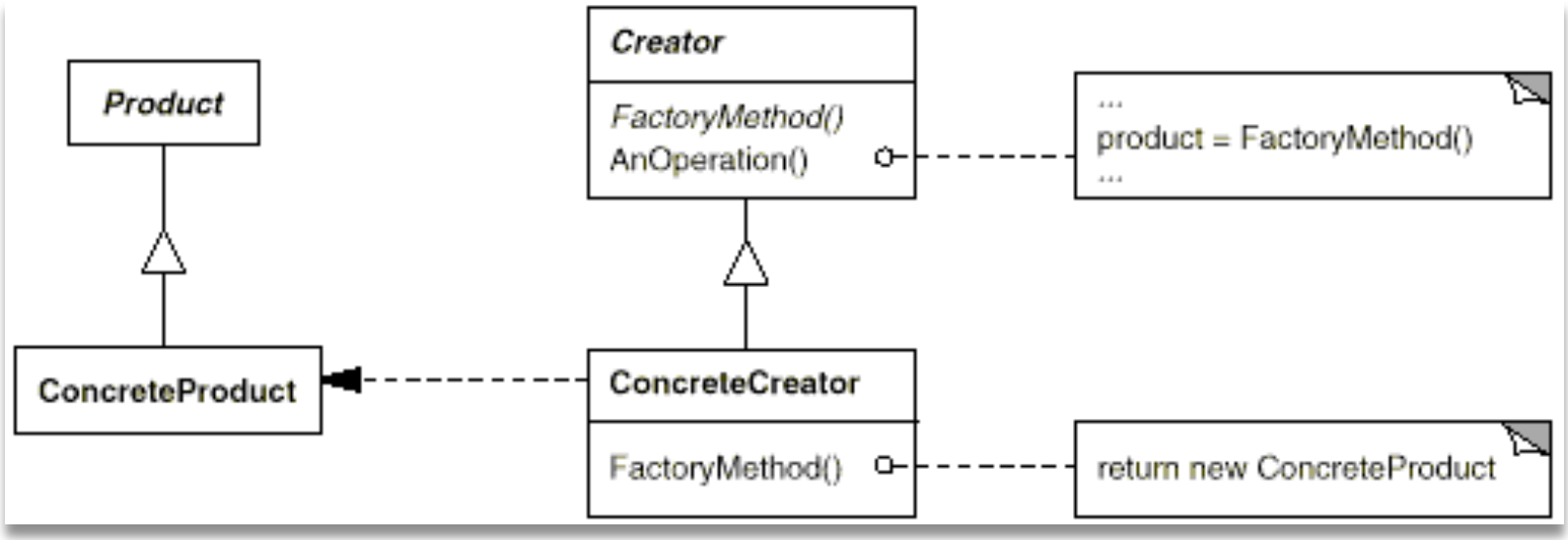
The Factory Method defines an interface for creating objects, but lets subclasses decide which classes to instantiate.

Injection molding presses demonstrate this pattern. Manufacturers of plastic toys process plastic molding powder, and inject the plastic into molds of the desired shapes [15]. The class of toy (car, action figure, etc.) is determined by the mold.

Factory Method Software Example

- JDBC DriverManager.getConnection

Factory Method Pattern UML



Prototype Pattern Description

- Intent: Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.
- AKA: N/A
- Motivation: A pet store application needs an easy way to add a litter of puppies. Different litters come from different parents and have different characteristics. Instead of creating object hierarchies to define these differences, you could adopt a cloning strategy along with parameterized types.
- Applicability:
  - When the classes to instantiate are specified at run-time, for example, by dynamic loading
  - to avoid building a class hierarchy of factories that parallels the class hierarchy of products
  - When instances of a class can have one of only a few different combinations of state

Prototype Real World Example

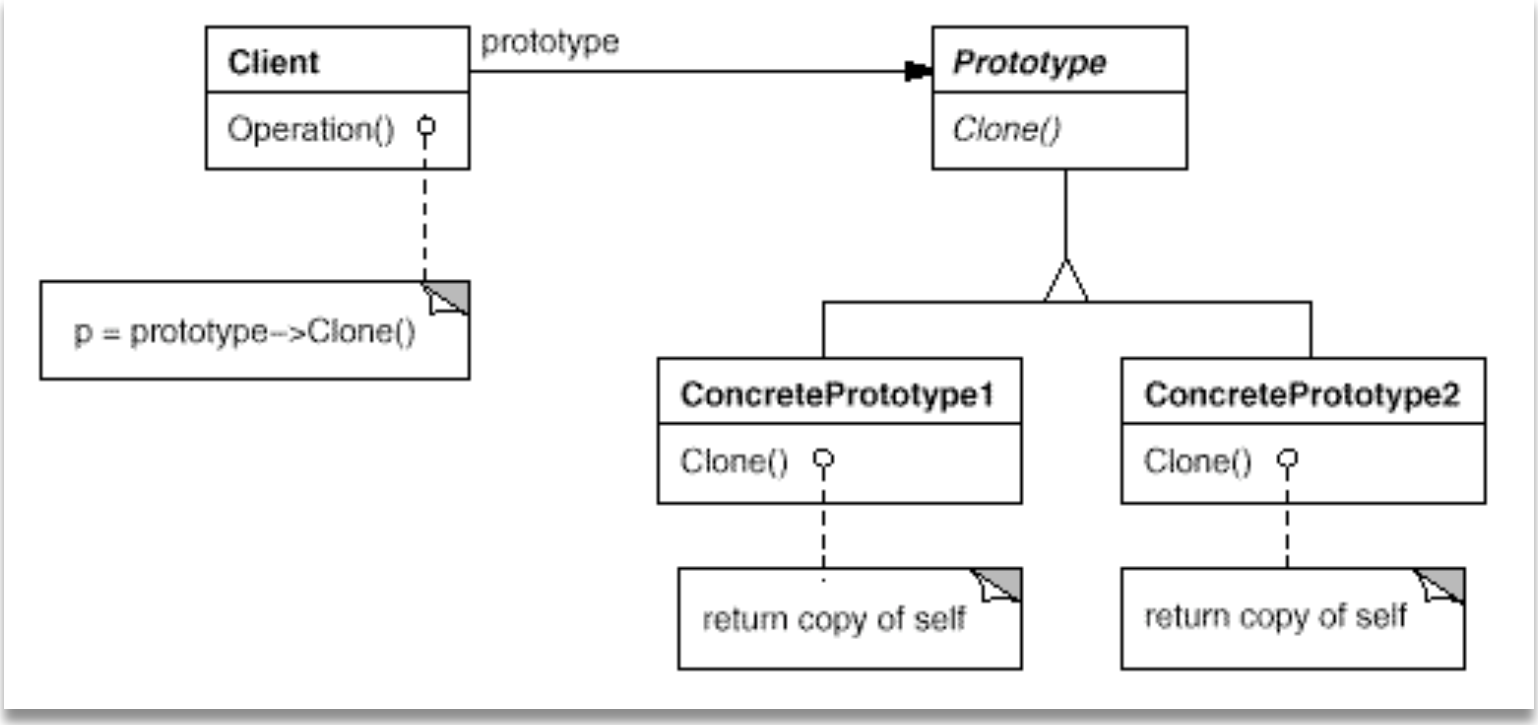
The Prototype pattern specifies the kind of objects to create using a prototypical instance.

Prototypes of new products are often built prior to full production, but in this example, the prototype is passive, and does not participate in copying itself. The mitotic division of a cell, resulting in two identical cells, is an example of a prototype that plays an active role in copying itself and thus, demonstrates the Prototype pattern. When a cell splits, two cells of identical genotype result. In other words, the cell clones itself.

Prototype Software Example

- The clone method in Java

Prototype 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 Real World Example

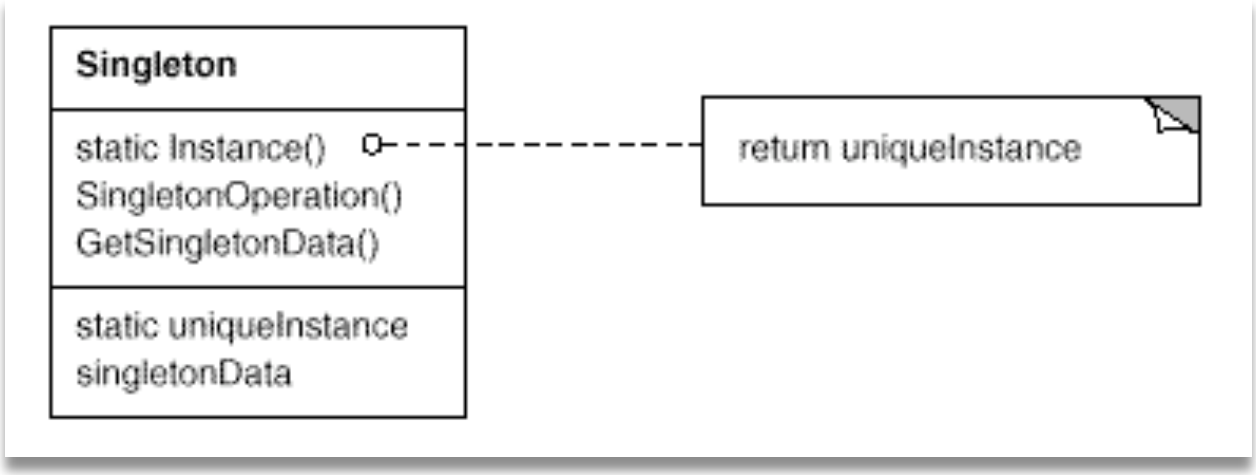
The Singleton pattern ensures that a class has only one instance, and provides a global point of access to that instance.

The Singleton pattern is named after the singleton set, which is defined to be a set containing one element. The office of the President of the United States is a Singleton. The United States Constitution specifies the means by which a president is elected, limits the term of office, and defines the order of succession. As a result, there can be at most one active president at any given time. Regardless of the personal identity of the active president, the title, "The President of the United States" is a global point of access that identifies the person in the office

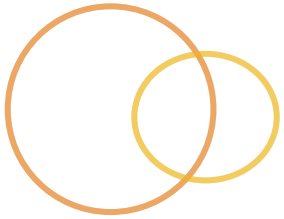
Singleton Software Example

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

Singleton Pattern UML



Summary

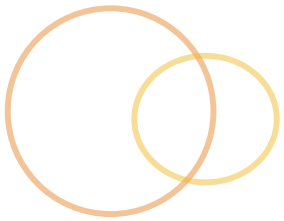


- Abstract Factory is used to encapsulate the creation of a Factory, based on different rules and knowledge
- Builder focuses on creating an object step by step without the caller knowing it
- Factory Method encapsulates the creation of a composition of an object through a method
- Prototype creates new objects by cloning objects that act as templates
- Singleton ensures only one instance of an object is loaded in memory

About DevelopIntelligence

- Founded in 2003
- Provides outsourced services to learning organizations in area of software development
- Represents over 35 years of combined experience, enabling software development community through educational and performance services
- Represents over 50 years of combined software development experience
- Delivered training to over 40,000 developers worldwide

Areas of Expertise



- Instruction
  - Java
  - J2EE
  - WebServices / SOA
  - Web Application Development
  - Database Development
  - Open Source Frameworks
  - Application Servers
- Courseware
  - Java Application Development
  - Java Web App Development
  - Enterprise Java Development
  - OOAD / UML
  - IT Managerial
  - Emerging Technologies and Frameworks

Contact Us

- For more information about our services, please contact us:
  - Kelby Zorgdrager
  - Kelby@DevelopIntelligence.com
  - 303-395-5340