# Creational pattern

From Wikipedia, the free encyclopedia

In software engineering, **creational design patterns** are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation. The basic form of object creation could result in design problems or in added complexity to the design. Creational design patterns solve this problem by somehow controlling this object creation.

Creational design patterns are composed of two dominant ideas. One is encapsulating knowledge about which concrete classes the system uses. Another is hiding how instances of these concrete classes are created and combined.[1]

Creational design patterns are further categorized into Object-creational patterns and Class-creational patterns, where Object-creational patterns deal with Object creation and Class-creational patterns deal with Class-instantiation. In greater details, Object-creational patterns defer part of its object creation to another object, while Class-creational patterns defer its object creation to subclasses.[2]

Five well-known design patterns that are parts of creational patterns are the

- Abstract factory pattern, which provides an interface for creating related or dependent objects without specifying the objects' concrete classes.[3]
- Builder pattern, which separates the construction of a complex object from its representation so that the same construction process can create different representations.
- Factory method pattern, which allows a class to defer instantiation to subclasses.[4]
- Prototype pattern, which specifies the kind of object to create using a prototypical instance, and creates new objects by cloning this prototype.
- Singleton pattern, which ensures that a class only has one instance, and provides a global point of access to it.[5]

## Contents



## Definition

The creational patterns aim to separate a system from how its objects are created, composed, and represented. They increase the system's flexibility in terms of the what, who, how, and when of object creation.[6]

## Usage

As modern software engineering depends more on object composition than class inheritance, emphasis shifts away from hard-coding behaviors toward defining a smaller set of basic behaviors that can be composed into more complex ones.[7] Hard-coding behaviors are inflexible because they require overriding or re-implementing the whole thing in order to change parts of the design. Additionally, hard-coding does not promote reuse and is

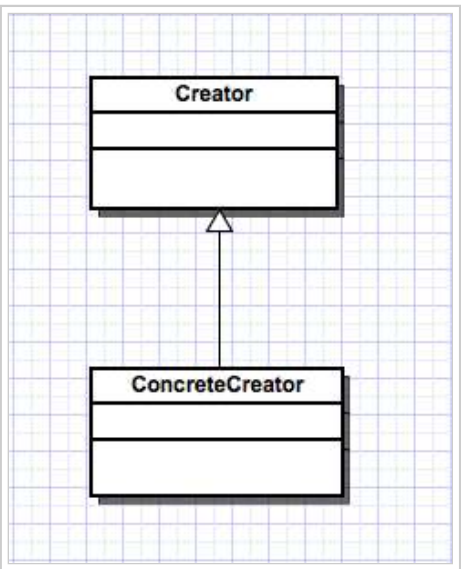
hard to keep track of errors. For these reasons, creational patterns are more useful than hard-coding behaviors. Creational patterns make design become more flexible. They provide different ways (patterns) to remove explicit references in the concrete classes from the code that needs to instantiate them.[8] In other words, they create independency for objects and classes.

Consider applying creational patterns when:

- A system should be independent of how its objects and products are created.
- A set of related objects is designed to be used together.
- Hiding the implementations of a class library or product, revealing only their interfaces.
- Constructing different representation of independent complex objects.
- A class wants its subclass to implement the object it creates.
- The class instantiations are specified at run-time.
- There must be a single instance and client can access this instance at all times.
- Instance should be extensible without being modified.

# Structure

Below is a simple class diagram that most creational patterns have in common. Note that different creational patterns require additional and different participated classes.



Creational Pattern class diagram.

**Participants**:

- **Creator**: Declares object interface. Returns object.
- **ConcreteCreator**: Implements object's interface.

# Examples

Some examples of creational design patterns include:

- Abstract factory pattern: centralize decision of what factory to instantiate
- Factory method pattern: centralize creation of an object of a specific type choosing one of several implementations
- Builder pattern: separate the construction of a complex object from its representation so that the same construction process can create different representations
- Lazy initialization pattern: tactic of delaying the creation of an object, the calculation of a value, or some other expensive process until the first time it is needed
- Object pool pattern: avoid expensive acquisition and release of resources by recycling objects that are no longer in use
- Prototype pattern: used when the type of objects to create is determined by a prototypical instance, which is cloned to produce new objects
- Singleton pattern: restrict instantiation of a class to one object

# See also

- Constructor
- Behavioral pattern
- Concurrency pattern
- Structural pattern

# References

1. Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). *Design Patterns*. Massachusetts: Addison-

Wesley. p. 81. ISBN 978-0-201-63361-0. Retrieved 2015-05-22.
2. Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). *Design Patterns*. Massachusetts: Addison-Wesley. ISBN 978-0-201-63361-0. Retrieved 2015-05-22.
3. Freeman, Eric; Freeman, Elisabeth; Sierra, Kathy; Bates, Bert (2004). Hendrickson, Mike; Loukides, Mike, eds. *Head First Design Patterns*. California: O'Reilly Media. p. 156. ISBN 978-0-596-00712-6. Retrieved 2015-05-22.
4. Freeman, Eric; Freeman, Elisabeth; Sierra, Kathy; Bates, Bert (2004). Hendrickson, Mike; Loukides, Mike, eds. *Head First Design Patterns*. California: O'Reilly Media. p. 134. ISBN 978-0-596-00712-6. Retrieved 2015-05-22.
5. Freeman, Eric; Freeman, Elisabeth; Sierra, Kathy; Bates, Bert (2004). Hendrickson, Mike; Loukides, Mike, eds. *Head First Design Patterns*. California: O'Reilly Media. p. 177. ISBN 978-0-596-00712-6. Retrieved 2015-05-22.
6. Judith, Bishop (2007). *C# 3.0 Design Patterns*. California: O'Reilly Media. p. 336. ISBN 978-0-596-52773-0. Retrieved 2015-05-22.
7. Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). *Design Patterns*. Massachusetts: Addison-Wesley. p. 84. ISBN 978-0-201-63361-0. Retrieved 2015-05-22.
8. Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). *Design Patterns*. Massachusetts: Addison-Wesley. p. 85. ISBN 978-0-201-63361-0. Retrieved 2015-05-22.