From Wikipedia

A software design pattern

is a general, reusable solution to a commonly occurring problem within a given context in software design.

It is not a finished design that can be transformed directly into source or machine code.

It is a description or template for how to solve a problem that can be used in many different situations.

Design patterns are formalized best practices that the programmer can use to solve common problems when designing an application or system.

Design patterns gained popularity in computer science after the book Design Patterns: Elements of Reusable Object-Oriented Software was published in 1994 by the so-called "Gang of Four" (Gamma et al.), which is frequently abbreviated as "GoF".

- *Gamma, Erich; Helm, Richard; Johnson, Ralph; Vlissides, John (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. ISBN 0-201-63361-2.*

The book is divided into two parts, with the first two chapters exploring the capabilities and pitfalls of object-oriented programming, and the remaining chapters describing 23 classic software design patterns.

The original publication date of the book was October 21, 1994 with a 1995 copyright, hence it is often cited with a 1995-year, despite being published in 1994.

The book was first made available to the public at the OOPSLA meeting held in Portland, Oregon, in October 1994.

In 2005 the ACM SIGPLAN awarded that year's Programming Languages Achievement Award to the authors, in recognition of the impact of their work "on programming practice and programming language design".

As of March 2012, the book was in its 40th printing

Chapter 1 is a discussion of object-oriented design techniques, based on the authors' experience, which they believe would lead to good object-oriented software design, including:

"Program to an 'interface', not an 'implementation'." (Gang of Four 1995:18)

Composition over inheritance: "Favor 'object composition' over 'class inheritance'." (Gang of Four 1995:20)

The authors claim the following as advantages of interfaces over implementation:

clients remain unaware of the specific types of objects they use, as long as the object adheres to the interface

clients remain unaware of the classes that implement these objects; clients only know about the abstract class(es) defining the interface

Use of an interface also leads to dynamic binding and polymorphism, which are central features of object-oriented programming.

The authors refer to inheritance as white-box reuse, with white-box referring to visibility, because the internals of parent classes are often visible to subclasses. In contrast, the authors refer to object composition (in which objects with well-defined interfaces are used dynamically at runtime by objects obtaining references to other objects) as black-box reuse because no internal details of composed objects need be visible in the code using them.