# Overview:

* The Template Method defines the skeleton of an algorithm in an operation, deferring some steps to subclasses:
  + Let us subclasses redefine certain steps of an algorithm without changing the algorithm’s structure.
  + Helps us generalize a common process, at an abstract level, from a set of different procedures.
* All about creating a template for an algorithm:
  + A template is just a method that defines an algorithm as a set of steps.
  + One or more of these steps is defined to be abstract (a method stub) and implemented by a subclass.
    - Ensures the algorithm’s structure stays unchanged, while subclasses provide some part of the implementation.
  + Enables you to define the sequence of steps and then override those steps that need to change
  + Software reuse is the fundamental goal of this method
    - This is why the pattern s used in many class libraries and framework development.
* One of the more common patterns in use today.

# Examples:

* In the real world templates are used all the time:
  + For architectural plans, and throughout the engineering domain
  + A template plan may be defined which is then built on with further variations.
* Suppose we want to make pizza:
  + The basic mechanism is the same, but extra materials are added based upon the customer’s choice
    - Whether he/she wants a vegetarian pizza or a non-vegetarian pizza.
* For an engineering student, in general, most of the subjects in the first semester are common for all concentrations.
  + Later, additional papers are added in his/her course based on his/her specialization (Computer Science, Electronics, etc.)
* We suppose we want to provide an algorithm to build a house:
  + The steps need to be performed to build a house are building foundation, building pillars, building walls and windows.
  + The important point is that the we cannot change the order of execution.
    - We cannot build windows before building the foundation.
  + Therefore, in this case we can create a template method that will use different methods to build the house.
    - Building the foundation for a house is same for all type of houses, whether it is a wooden house or a glass house.
    - We can provide a base implementation for this, if subclasses want to override this method, they can but mostly it is common for all the types of houses.
* Another example would be the methods for connecting and querying Oracle or SQL Server databases.
  + The methods may be different but they share the same conceptual process.
* Example in the JDK include:
  + Non-abstract methods of java.io.InputStream, java.ioOutputStream, java.io.Reader and java.io.Writer.
  + Non-abstract methods of java.util.AbstractList, java.util.AbstractSet and java.util.AbstractMap.
  + The Arrays class uses if for sorting, JFrame uses update () as a template method.
  + Subclasses of the JFrame use paint (Graphics graphic) as their hook method.

# Often used in frameworks

* This pattern shows up so often because it is a great design tool for creating frameworks.
  + The framework controls how something is done.
  + Leaves he person using the framework to specify your own details about what is actually happening at each step of the framework’s algorithm.
* Consider an application framework that provides Application and Document classes.
  + The Application class is responsible for opening existing documents stored in an external format, such as a file.
  + A Document object represents the information in a document once it is read from the file.
* Applications built with the framework can subclass Application and Document to suit specific needs.
  + A drawing application defines DrawApplication and DrawDocument subclasses.
  + A spreadsheet application defines Spreadsheet-Application and Spreadsheet-Document subclasses.

# When to use this pattern?

* When you need to support multiple algorithms that behave conceptually the same but have different implementations for each of their steps.
* When you want to avoid code duplication in an algorithm
  + Implement variations of the algorithm in subclasses
  + A good example of “refactoring to generalize”
  + First, identify the differences in the existing code and then separate the differences into new operations.
    - You then replace the differing code with a template method that calls one of these new operations.
* When you want to control at what points sub classing is allowed:
  + Define a template method that calls “hook” operations at specific points, thereby permitting extensions only at those points.
* When behavior of an algorithm can vary, you let subclasses implement the behavior through overriding.

# Implementation:

* The Template Method Pattern is a very common pattern:
  + There are many implementations of the template methods that do not quite look like the textbook design of the pattern.
  + It will take practice to identify when and how to use this pattern.
* Most of the time, subclasses call methods in the super class
  + In the template pattern, the superclass template method calls methods (abstract in base class) in the subclasses.
  + Known as the Hollywood Principle – “don’t’ call us, we’ll call you”.
* The Hollywood Principle guides s to put decision making in high-level modules that can decide how and when to call low-level modules.
* The main template method is in the base class which is an abstract class.
  + The abstract class may define concrete methods, abstract methods, and hooks
  + Abstract methods are implemented by the subclasses.
* A hook is a method that is declared in the abstract class
  + Given an empty or default implementation
  + Gives subclasses the ability to “hook into” the algorithm at various points
  + A subclass is also free to ignore the hook.

# Participants:

* **AbstractClass:**
* Contains the template method and abstract versions of the operations used in the template method.
  + The template method defines the skeleton of an algorithm.
* The template method makes se of the primitive operations to implement an algorithm
  + Calls primitive operations as well as operations define in the AbstractClass or those of other objects.
  + Decoupled from the actual implementation of these operations.
* **ConcreteClass:**
  + Implements the abstract operations, which are called when the TempalteMethod () needs them.
  + May be many concrete classes, each implementing the full set of operations required by the template method.

# Implementation issue:

* The primitive operations that a template method calls can be declared as protected methods
  + Ensures that they are only called by the template method.
* The template method itself should not be overridden.
  + To prevent subclasses from changing the algorithm, declare the template method as final.
* An important goal in designing template methods is to minimize the number of abstract operations used to flesh out the algorithm.
  + We do not have a choice, overriding is a must in this case
* To reuse an abstract class effectively, subclass writers must understand which operations are designed for overriding.
  + The more operations that need overriding, the more tedious things get for clients.