# IN710 Object Oriented Systems Development

# Semester 1, Design Patterns Cheat Sheet 2019

## Abstract Factory

* **Definition:** Provides interfaces for creating families of related objects without specifying their concrete classes.
* **Advantages:** Helps control the classes of objects that an application creates. Encapsulates the responsibility and process of creating product objects, isolating clients from implementation classes.
* **Disadvantages:** Increases overall code complexity by creating additional classes.

## Adapter

* **Definition:** Converts the interface of a class into another interface that the client expects. Let’s classes work together that couldn’t otherwise because of incompatible interfaces.
* **Advantages:** Helps reusability and flexibility. Can use polymorphism to swap between different implementations of adapters.
* **Disadvantages:** Requests are forwarded, so there is a slight increase in the overhead.

## Builder

* **Definition:** Builds a complex object using simple objects and a step by step approach.
* **Advantages:** The constructor’s parameters are reduced and are provided in highly readable method calls.
* **Disadvantages:** An increase in the number of lines of code.

## Command

* **Definition:** A request is wrapped under an object as command and passed to invoker object. Invoker object looks for the appropriate object which can handle this command and passes the command to the corresponding object.
* **Advantages:** Extensible code as new commands can be added without changing the existing code.
* **Disadvantages:** An increase in the number of classes for each individual command.

## Decorator

* **Definition:** Allows new functionality to be added to an existing object without altering its structure.
* **Advantages:** An alternative to sub-classing. Adds behaviour at compile time, and the change affects all instances of the original class.
* **Disadvantages:** May result in many small objects in our design, and overuse can be complex.

## Observer

* **Definition:** Used when there is a one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically.
* **Advantages:** Provides a low coupled design between objects that interact and flexible with requirement changes.
* **Disadvantages:** Memory leaks due to explicit registering and unregistering of observers.

## Simple Factory

* **Definition:** Defines an interface for creating an object, but lets the classes that implement the interface decide which class to be instantiated.
* **Advantages:** Hides concrete classes from the client. The client doesn’t know the implementation in the concrete class because it is accessed through the interface.
* **Disadvantages:** Refactoring an existing class to use factories breaks existing clients.

## Singleton

* **Definition:** Involves a single class which is responsible for creating an object while making sure that only a single object is created.
* **Advantages:** Saves memory because the object is not created at each request. Only a single instance is reused.
* **Disadvantages:** Dependencies are hidden in the code instead of exposing them through interfaces.

## Strategy

* **Definition:** Creates objects which represent various strategies and a context object whose behaviour varies as per its strategy object.
* **Advantages:** The application can switch strategies at runtime and enables the client to choose the required algorithm, without using a “switch” statement or a series of “if-else” statements.
* **Disadvantages:** The application must be aware of all strategies to select the right one for the appropriate situation.

## Template

* **Definition:** An abstract class which exposes defined templates to execute it methods.
* **Advantages:** Uses inheritance and not composition. Only a few methods need to be overridden.
* **Disadvantages:** A method may be implemented that shouldn’t or not implementing an abstract method at all.