https://www.javatpoint.com/design-patterns-in-java
https://sourcemaking.com/design_patterns
https://www.tutorialspoint.com/design_pattern/index.htm

Design Pattens - a general **repeatable solution to a commonly occurring problem** in software design
-**a** description or **template for how to solve a problem** that can be **used in many different situations**

1. Behavioral Pattern
-explains **how objects interact.**
-It describes how different objects and classes send messages to each other to make things happen and
-how the steps of a task are divided among different objects.
-Where Creational patterns mostly describe a moment of time (the instant of creation), and Structural patterns describe a more or less static structure, Behavioral patterns describe a process or a flow

BEHAVIORAL DESIGN PATTERNS
Chain of responsibility - A way of passing a request between a chain of objects to handle request
Command - Encapsulate a command request as an object and send to invoker.
Invoker object looks for the appropriate object which can handle this command and pass the command to the corresponding object
Interpreter - A way to evaluate language grammar or expression.
Involves implementing an expression interface which tells to interpret a particular context.
This pattern is used in SQL parsing, symbol processing engine etc.
Iterator - Sequentially access the elements of a collection
Mediator - Define an object that encapsulates how a set of objects interact.
Memento - Restore state of an object to a previous state.
Null Object - Designed to act as a default value of an object
Observer - A way of notifying change to a number of classes
State - Alter an object's behavior when its state changes
Strategy - Encapsulates an algorithm inside a class
Template method - Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.
Visitor - Define a new operation without changing the classes of the elements on which it operates.

2. **Creational patterns** are design patterns that deal with object creation mechanisms and are used in situations when basic form of object creation could result in design problems or increase complexity of a code base.

CREATIONAL DESIGN PATTERNS
Abstract Factory - Creates an instance of several families of classes
Builder - construct a complex object from simple objects using step-by-step approach
Factory Method - Define an interface for creating an object, but let subclasses decide which class to instantiate.
Object Pool - used to manage the object caching.
Basically, an Object pool is a container which contains a specified amount of objects.
When an object is taken from the pool, it is not available in the pool until it is put back.
Recycle objects that are no longer in use
Prototype - A fully initialized instance to be copied or cloned
Singleton - A class of which only a single instance can exist

3.**Structural design** patterns are design patterns that ease the design by
-identifying a simple way to realise **relationships between entities** or defines a manner
-for creating relationships between objects.

STRUCTURAL DESIGN PATTERNS
Adapter - Match interfaces of different classes
Bridge - Separates an object's interface from its implementation
Composite - A tree structure of simple and composite objects
Decorator - Add responsibilities to objects dynamically
Facade - A single class that represents an entire subsystem
Flyweight - A fine-grained instance used for efficient sharing
Private Class Data - Restricts accessor/mutator access
Proxy - An object representing another object

CREATIONAL

These design patterns are all about class instantiation. This pattern can be further divided into class-creation patterns and object-creational patterns. While class-creation patterns use inheritance effectively in the instantiation process, object-creation patterns use delegation effectively to get the job done.
Manage class selection and object creation. Maybe delegate object creation, or restrict object instances.

Creational Singleton - A class of which only a single instance can exist, or given at a time.
Intent - Ensure a class has only one instance, and provide a global point of access to it. AND Encapsulated "just-in-time initialization" or "initialization on first use".
All methods are accessed through the singleton.