1. Three groups of design patters
    - Creational
    - Structural
    - Behavioral

2. Design patterns

Well tested solutions to common problems. They are best practices on how classes and objects might be arranged to accomplish a result.

3. Creational Patterns
    - Deals with creation of objects
    - Abstract Factory
    - Builder
    - Factory
    - Prototype
    - Singleton

4. Abstract Factory
    - Allows you to create families of related objects without specifying a concrete class
    - Use when you have many object that can be added or changed dynamically during runtime

5. Builder
    - Pattern used to create objects made from other objects
    - When you want the creation of these parts to be independent of the main object

6. Factory
    - Allows you to create objects without specifying the exact class of objects that will be created.
    - When a method returns one of several possible classes that share a common superclass.

7. Prototype

Creating new objects by cloning(copying) other objects.

8. Singleton
    - Pattern used when you want to instantiate only one object of a class.
    - I uses lazy instantiation which means if the object isn't needed it isn't created.

9. Structural

Deals with how classes are designed. How things like inheritance, composition and aggregation can be used to provide extra functionality.

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

10. Adapter
    - Allows 2 incompatible interfaces to work together.
    - Used when the client expects a (target) interface
    - The Adapter class allows the use of the available interface and the Target interface.

11. Bridge
    - Decouple an abstraction from its implementation so that the two can vary independently
    - When to use it? When you want to be able to change both the abstractions (abstract classes) and concrete classes independently

12. Composite
    - Allows you to treat individual objects and compositions of objects uniformly
    - They allow you to represent part-whole hierarchies
    - You can structure data, or represent the inner workings of every part of a whole object individual

13. Decorator
    - The Decorator allows you to modify an object dynamically
    - You could use it when you want the capabilities of inheritance with subclasses, but you need to add functionality at run time
    - It is more flexible than inheritance

14. Facade
    - When you create a simplified interface that performs many other actions behind the scenes
    - Can I withdraw $50 from the bank?
    - Check if the checking account is valid
    - Check if the security code is valid
    - Check if funds are available
    - Make changes accordingly

15. Flyweight
    - Used when you need to create a large number of similar objects (> 100K)
    - To reduce memory usage, you share Objects that are similar in some way rather than creating new ones
    - Intrinsic State: Color
    - Extrinsic State: Size

16. Proxy
    - Provide a class which will limit access to another class
    - You may do this for security reasons, because an Object is intensive to create, or is accessed from a remote location.

17. Behavioral
    - Specifically concerned with communication between objects as the program is running:
    - Chain of Responsibility
    - Command
    - Interpreter
    - iterator
    - Memento
    - Observer
    - State
    - Strategy
    - Template Method
    - Visitor

18. Chain of Responsibility

This pattern sends data to an object and if that object can't use it, it sends it to any number of other objects that may be able to use it

19. Command
    - Pattern in which an object is used to represent and encapsulate all the information needed to call a method at a later time.
    - This information includes the method name, the object that owns the method and values for the method parameters
    - Allows you to store lists of code that is executed at a later time or many times.

20. Interpreter
    - Useful when used with Java Reflection
    - It is used to convert one representation of data into another

21. Iterator
    - The Iterator pattern provides you with a uniform way to access different collections of Objects
    - If you get an Array, ArrayList and Hashtable of Objects, you pop out an iterator for each and treat them the same
    - This provides a uniform way to cycle through different collections

22. Memento
    - A way to store previous states of an Object easily
    - Memento: The basic object that is stored in different states
    - Originator: Sets and Gets values from the currently targeted Memento. Creates new Mementos and assigns current values to them

23. Observer
    - When you need many other objects to receive an update when another object changes.
    - Loose coupling is a benefit. The Subject (publisher) doesn't need to know anything about the Observers(subscribers)

24. State
    - Allows an object to alter its behavior when its internal state changes. The object will appear to change its class
    - Context (Account): Maintains an instance of a ConcreteState subclass that defines the current state
    - State: Defines an interface for encapsulating the behavior associated with a particular state of the Context
    - Concrete State: Each subclass implements a behavior associated with a state of Context

25. Strategy
    - When you want to define a class that will have one behavior that is similar to other behaviors in a list.
    - When you need to use one of several behaviors dynamically

26. Template
    - Used to create a group of subclasses that have to execute a similar group of methods
    - You create an abstract class that contains a method called the Template Method
    - The Template Method contains a series of method calls that every subclass object will call
    - The subclass objects can override some of the method calls
    -

27. Visitor
    - Allows you to add methods to classes of different types without much altering to those classes
    - You can make completely different methods depending on the class used