Решения за многократна употреба на често срещани проблеми
(Java базирани примери)

Решенията за многократна употреба на често срещани проблеми в контекста на софтуерния дизайн се базират на т.нар. дизайн шаблони.

Делят на три групи:
- **Creational** – за създаване
- **Structural** – структурни
- **Behavioral** – поведенчески

1. **Шаблони за създаване (Creational Patterns)** – осигуряват начин да се създават обекти на класове, скривайки логиката по създаването им, вместо да се инстанцират директно чрез оператора им. Примери: Factory, Abstract Factory, Builder, Singleton, Prototype.

1.1. Factory
Използва се най-често, когато искаме да създаваме обекти на някакъв клас без да излагаме на външния свят самата логика по създаването му. Имаме някакви причини да искаме да я държим енкапсулирана вътре в имплементацията на нашия клас.
Използва се когато имаме родителски клас и няколко наследника и искаме да създаваме един от наследниците на родителския клас според подаден параметър.
Примери от JDK-то: valueOf( ) метода на wrapper класовете като Boolean, Integer и т.н., of( ) методите на List, Set, Map и т.н., of( ) метода на Path, of( ) метода на Stream.

1.2. Builder
Решава някои проблеми на Factory шаблона за класове с много атрибути, някои от които са опционални. Примери от JDK-то: StringBuilder, StringBuffer, HttpClient, HttpRequest.
Имплементация: създаваме вложен публичен статичен клас с името на оригиналния клас и конкатенирана думичката Builder и копираме всички параметри от външния клас в Builder класа. Оригиналния клас трябва да има private конструктор, който приема инстанция на вложения Builder клас. Builder класът трябва да има публичен конструктор с всички задължителни атрибути като параметри. setter методи за всички опционални параметри, които връщат същата Builder инстанция. build метод, който връща обекта this (оригиналния клас).

1.3. Singleton
Представлява клас, от който може да съществува най-много една инстанция. Може да се модифицира да е с нетърпеливо или мързеливо инстанциране или да е устойчив на множество нишки
Имплементация: private constructor. private static член променлива от същия клас, която реферира единствената инстанция на класа. public static метод, който връща инстанцията на класа.
Типични употреби: logging, caching, thread pools, analyzer класа от второто домашно (в някакъв shared service, който няма state), в други дизайнерски шаблони: Factory, Builder, Facade, Prototype

2. **Структурни шаблони (Structural Patterns)** – осигуряват различни начини за създаване на по-сложни класове чрез наследяване и композиция от по-прости класове.
Примери: Adapter, Composite, Proxy, Flyweight, Facade, Bridge, Decorator.

2.1. Flyweight (Structural)
- Позволява да се съберат повече обекти в наличната памет чрез споделяне на общите части на state-а им
- Намалява memory footprint-а на програмата
- Може да подобри бързодействието в приложения, където инстанцирането на обектите е скъпа операция
- flyweight обектите са immutable: всяка операция, която променя състоянието им трябва да се изчислява от factory-то

- Примери от JDK-то: String с имплементацията на String Pool-а, Integer.valueOf(int), Byte.valueOf(byte) и подобните са останалите wrapper типове.
- Имплементация:
  - интерфейс, който дефинира операциите, които клиентския код може да извършва върху flyweight обектите
  - една или повече конкретни имплементации на този интерфейс
  - factory, което отговаря за инстанциране и кеширане

3. **Поведенчески шаблони (Behavioral Patterns)** – свързани са с комуникацията между обекти. Примери: Template Method, Command, Mediator, Chain of Responsibility, Observer, Strategy, State, Visitor, Interpreter, Iterator, Memento.

3.1. Iterator

- Позволява последователно обхождане на колекция от обекти.
- Примери от JDK-то: java.util.Iterator и обхождането на колекции

3.2. Command

- Използва се за имплементиране на loose coupling в модел тип заявка – отговор
- Пример от JDK-то: java.lang.Runnable

3.3. Observer

- Когато искаме в някакви наши класове да следим за промяна в инстанция на друг клас. Удобен е, когато се интересуваме от състоянието на даден обект и искаме да бъдем информирани, когато има промяна в състоянието.
- Обектът, който наблюдава за промяна на състоянието на друг обект, се нарича Observer, а наблюдаваният обект се нарича Subject
- Примери от JDK-то: java.util.Observer, java.util.Observable

3.4. Strategy

- Прилага се, когато имаме множество алгоритми за дадена задача и клиентът решава по кое време на изпълнение, коя имплементация на алгоритъма да се ползва
- Примери в JDK-то: Collections.sort( ), който сортира по различен критерий/ алгоритъм в зависимост от подадения Comparator