Design Patterns

Creational **Structural** **Behavioral**

Gang of Four • 23 klasična obrasca • Java primjeri

Povijest i Gang of Four

🗓️ 1977 - POČETAK

Christopher Alexander objavljuje "A Pattern Language"
253 uzorka za arhitekturu zgrada

📚 1994 - REVOLUCIJA

"Design Patterns: Elements of Reusable OO Software"
Prodano 500.000+ primjeraka, 13 jezika

👥 GANG OF FOUR

- Erich Gamma - IBM, Eclipse, VS Code
- Richard Helm - IBM, Software Architecture
- Ralph Johnson - University of Illinois
- John Vlissides - IBM Research (1961-2005)

Zašto su *design patterni* važni?

87%

senior developera koristi

73%

tech intervjua uključuje

45%

brže održavanje koda

🎯 KLJUČNE PREDNOSTI

- Zajednički jezik - tim razumije "koristi Strategy" bez objašnjavanja
- Reusability - provjerena rješenja, ne izmišljaj toplu vodu
- Skalabilnost - kod lakše proširiti i modificirati
- Testabilnost - loose coupling = lakše unit testiranje

Izvor: Stack Overflow Developer Survey 2024

23 GoF Design Patterns

CREATIONAL (5)

Kreiranje objekata

- Abstract Factory
- Builder
- Factory Method
- Prototype
- Singleton

STRUCTURAL (7)

Kompozicija klasa

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

BEHAVIORAL (11)

Komunikacija

- Chain of Resp.
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template
- Visitor

CREATIONAL

Abstract Factory Pattern

```
// Sučelja proizvoda
interface Button { void render(); }
interface Checkbox { void check(); }

// Abstract Factory
interface GUIFactory {
    Button createButton();
    Checkbox createCheckbox();
}

// Concrete Factory - Windows
class WindowsFactory implements GUIFactory {
    Button createButton() {
        return new WinButton();
    }
    Checkbox createCheckbox() {
        return new WinCheckbox();
    }
}
```

📌 KADA KORISTITI?

- Puno povezanih objekata
- Cross-platform UI (Win/Mac/Linux)
- Konzistentnost proizvoda

PRIMJERI U PRAKSI

- Java AWT - Toolkit klasa
- Swing - Look and Feel
- Spring - BeanFactory

Korištenje: 15% Java projekata • Kompleksnost: ⭐⭐⭐

CREATIONAL

Prototype Pattern

```
// Prototype sučelje
interface Prototype {
    Prototype clone();
}

// Concrete Prototype - Game enemy
class Enemy implements Prototype {
    private String name;
    private int health, level;

    public Prototype clone() {
        return new Enemy(name, health, level);
    }
}

// Korištenje - brzo kloniranje
Enemy orc = new Enemy("Orc", 100, 5);
Enemy clone = orc.clone(); // Brza kopija!
```

📌 KADA KORISTITI?

- „Skupo” kreiranje objekata (DB)
- Mnogo sličnih objekata (gaming)
- Dinamičko učitavanje klasa

PRIMJERI U PRAKSI

- Object.clone() - Java Cloneable
- Unity - Instantiate() za prefabs
- JavaScript - Object.assign()
- Spring - scope="prototype"

Korištenje: 12% Java projekata • Gaming: ⭐⭐⭐⭐⭐

STRUCTURAL

Proxy Pattern

```
// Zajedničko sučelje
interface Image { void display(); }

// Real Subject - skupo učitavanje
class RealImage implements Image {
    RealImage(String file) {
        loadFromDisk(file); // SPORO!
    }
}

// Proxy - lazy loading
class ImageProxy implements Image {
    private RealImage real;
    private String file;

    void display() {
        if (real == null)
            real = new RealImage(file);
        real.display();
    }
}
```

📌 KADA KORISTITI?

- Virtual - lazy loading
- Protection - access control
- Remote - mrežna komunikacija
- Caching - cache rezultata

PRIMJERI U PRAKSI

- Hibernate - Lazy loading entiteta
- Spring AOP - @Transactional proxy
- JPA - EntityProxy
- Java RMI - Remote proxy

Korištenje: 28% Java projekata • Enterprise: ⭐⭐⭐⭐⭐

STRUCTURAL

Composite Pattern

```
// Component - zajedničko sučelje
interface FileSystem { int getSize(); }

// Leaf - pojedinačni element
class File implements FileSystem {
    private int size;
    public int getSize() { return size; }
}

// Composite - sadrži djecu
class Folder implements FileSystem {
    List<FileSystem> children =
        new ArrayList<>();

    public int getSize() {
        return children.stream()
            .mapToInt(FileSystem::getSize)
            .sum();
    }
}
```

📌 KADA KORISTITI?

- Tree strukture (file system)
- Uniformno tretiranje lista i kompozita
- Rekurzivne operacije

PRIMJERI U PRAKSI

- Swing/AWT - Component/Container
- React - Component tree
- DOM - HTML element hijerarhija
- JSON/XML - Parsiranje

Korištenje: 22% Java projekata • UI frameworki: ⭐⭐⭐⭐⭐

BEHAVIORAL

Command Pattern

```
// Command interface
interface Command {
    void execute();
    void undo();
}

// Concrete Command - Spotify playlist
class AddToPlaylistCmd implements Command {
    private Playlist playlist;
    private Song song;

    void execute() { playlist.add(song); }
    void undo() { playlist.remove(song); }
}

// Invoker - history stack
class CommandHistory {
    Stack<Command> history = new Stack<>();
    void push(Command c) { c.execute(); history.push(c); }
    void undo() { history.pop().undo(); }
}
```

📌 KADA KORISTITI?

- Undo/Redo funkcionalnost
- Odgođeno izvršavanje (queue)
- Logging operacija
- Transakcije

PRIMJERI U PRAKSI

- Runnable - Java threading
- Redux - Actions
- Git - Commit kao command
- Text editori - Ctrl+Z

Korištenje: 24% Java projekata • Undo/Redo: ⭐⭐⭐⭐⭐

BEHAVIORAL

State Pattern

```
// State interface - video player
interface PlayerState {
    void play(VideoPlayer p);
    void pause(VideoPlayer p);
}

// Concrete State
class StoppedState implements PlayerState {
    void play(VideoPlayer p) {
        p.setState(new PlayingState());
    }
    void pause(VideoPlayer p) { /* ignore */ }
}

// Context
class VideoPlayer {
    private PlayerState state = new StoppedState();
    void setState(PlayerState s) { state = s; }
    void pressPlay() { state.play(this); }
}
```

📌 KADA KORISTITI?

- State machines (TCP, Orders)
- Zamjenjuje if/switch s polimorfizmom
- Čišći, proširiv kod

PRIMJERI U PRAKSI

- TCP Connection - states
- Order status - Pending/Shipped
- Game AI - Idle/Attack/Flee
- UI components

Korištenje: 18% Java projekata • State machines: ⭐⭐⭐⭐⭐

BEHAVIORAL

Iterator Pattern

```
// Iterator interface - Java built-in
interface Iterator<T> {
    boolean hasNext();
    T next();
}

// Concrete Iterator - Instagram feed
class FeedIterator implements Iterator<Post> {
    private List<Post> posts;
    private int index = 0;

    boolean hasNext() {
        return index < posts.size();
    }
    Post next() {
        return posts.get(index++);
    }
}

// Korištenje - for-each koristi Iterator!
for (Post p : feed) { display(p); }
```

📌 KADA KORISTITI?

- Uniformno prolaženje kolekcija
- Skriva internu strukturu
- Više iteratora paralelno
- Lazy evaluation moguć

PRIMJERI U PRAKSI

- java.util.Iterator - sve kolekcije
- Stream API - lazy iteration
- ResultSet - JDBC database
- Scanner - reading input

Korištenje: 95%+ Java projekata • NAJPOPULARNIJI PATTERN!

BEHAVIORAL

Chain of Responsibility

```
// Handler - support ticket sistem
abstract class SupportHandler {
    protected SupportHandler next;

    void setNext(SupportHandler h) {
        next = h;
    }

    abstract void handle(Ticket t);
}

// Concrete Handler
class BotSupport extends SupportHandler {
    void handle(Ticket t) {
        if (canHandle(t)) solve(t);
        else if (next != null) next.handle(t);
    }
}

// Lanac: Bot → Agent → Manager → CEO
```

📌 KADA KORISTITI?

- *Decouples* sender od receivera
- Fleksibilna obrada zahtjeva
- Dinamički lanac *handlera*

PRIMJERI U PRAKSI

- Servlet Filters - request proc.
- Spring Security - filter chain
- Logging - log level handlers
- DOM events - event bubbling

Korištenje: 20% Java projekata • Web frameworki: ⭐⭐⭐⭐⭐

BEHAVIORAL

Mediator Pattern

```
// Mediator - chat room primjer
interface ChatMediator {
    void sendMessage(String msg, User sender);
    void addUser(User user);
}

// Concrete Mediator
class ChatRoom implements ChatMediator {
    List<User> users = new ArrayList<>();

    void sendMessage(String msg, User sender) {
        users.stream()
            .filter(u -> u != sender)
            .forEach(u -> u.receive(msg));
    }
}

// Colleague - koristi mediator
class User { ChatMediator mediator; }
```

📌 KADA KORISTITI?

- Centralizira komunikaciju
- Bez Mediatora: N×(N-1) veza
- S Mediatorom: samo N veza

PRIMJERI U PRAKSI

- MVC - Controller kao mediator
- Event Bus - Guava EventBus
- Message Broker - Kafka, RabbitMQ
- Dialog windows - UI komponente

Korištenje: 15% Java projekata • Microservices: ⭐⭐⭐⭐

STRUCTURAL

Flyweight Pattern

```
// Flyweight - dijeljeno stanje (immutable)
class CharacterStyle {
    final String font;
    final int size;
    final Color color;
}

// Factory s cache-om
class StyleFactory {
    private Map<String, CharacterStyle> cache
        = new HashMap<>();

    CharacterStyle getStyle(String font, int size) {
        String key = font + size;
        return cache.computeIfAbsent(key,
            k -> new CharacterStyle(font, size));
    }
}

// Ušteda: 1M objekata → samo 100 stilova
```

📌 KADA KORISTITI?

- Intrinsic (dijeljeno) stanje
- Extrinsic (jedinstveno) stanje
- Masivna ušteda memorije

PRIMJERI U PRAKSI

- String.intern() - String pool
- Integer.valueOf() - cache -128..127
- Word processors - stilovi teksta
- Game engines - sprites, teksture

Korištenje: 6% Java projekata • Memory optimization: ⭐⭐⭐⭐⭐

STRUCTURAL

Bridge Pattern

```
// Implementor - platforma za slanje
interface NotificationSender {
    void send(String msg);
}
class EmailSender implements NotificationSender {...}
class SMSSender implements NotificationSender {...}

// Abstraction - tip notifikacije
abstract class Notification {
    protected NotificationSender sender; // BRIDGE!
    abstract void notify(String m);
}

// Refined Abstraction
class UrgentNotification extends Notification {
    void notify(String m) {
        sender.send("URGENT: " + m);
    }
}
```

📌 KADA KORISTITI?

- Odvaja apstrakciju od implementacije
- Bez Bridge: 3×3=9 klasa
- S Bridge: 3+3=6 klasa

PRIMJERI U PRAKSI

- JDBC - Driver/Connection
- SLF4J - Logging API/Impl
- Swing - UI/Platform
- Remote controls

Korištenje: 8% Java projekata • Decoupling: ⭐⭐⭐⭐

Visitor & Memento Patterns

VISITOR - Double Dispatch

```
interface FileVisitor {
    void visit(TextFile f);
    void visit(ImageFile f);
}

class CompressionVisitor
        implements FileVisitor {
    void visit(TextFile f) { gzip(f); }
    void visit(ImageFile f) { jpeg(f); }
}
```

Primjeri:
- java.nio.file.FileVisitor
- ANTLR - AST (Abstract Syntax Tree) visitors
- Eclipse JDT - ASTVisitor

MEMENTO - Snapshot

```
// Memento - čuva stanje
class EditorMemento {
    private final String content;
}

// Originator
class TextEditor {
    EditorMemento save() {
        return new EditorMemento(content);
    }
    void restore(EditorMemento m) {
        content = m.getContent();
    }
}
```

Primjeri:
- Text editori - Undo history
- Games - Save/Load
- Database - Transaction rollback

Visitor: 8% • Memento: 10% Java projekata

BEHAVIORAL

Interpreter Pattern

```
// Expression interface - AST čvorovi
interface Expression { int interpret(); }

// Terminal expression - broj
class NumberExpr implements Expression {
    private int number;
    public int interpret() { return number; }
}

// Non-terminal - operacija
class AddExpr implements Expression {
    private Expression left, right;

    public int interpret() {
        return left.interpret() +
               right.interpret();
    }
}

// Korištenje: 5 + 3
Expression e = new AddExpr(
    new NumberExpr(5), new NumberExpr(3));
e.interpret(); // = 8
```

📌 KADA KORISTITI?

- Parsiranje jednostavnih jezika
- DSL (Domain Specific Language)
- Regularni izrazi
- Matematički izrazi

PRIMJERI U PRAKSI

- java.util.regex - Pattern
- SpEL - Spring Expression Lang
- SQL parseri - JSqlParser
- ANTLR - Parser generator

Korištenje: 3% Java projekata • DSL: ⭐⭐⭐⭐

Patterni u Popularnim Frameworcima

SPRING

- Singleton - @Service
- Factory - BeanFactory
- Proxy - AOP
- Template - JdbcTemplate
- Observer - @EventListener

JAVA SDK

- Iterator - Collection
- Decorator - I/O Streams
- Strategy - Comparator
- Flyweight - Integer cache
- Builder - StringBuilder

REACT/JS

- Composite - Components
- Observer - useState
- HOC - Decorator
- Command - Redux
- Factory - createElement

HIBERNATE

- Proxy - Lazy loading
- Factory - SessionFactory
- State - Entity lifecycle
- Strategy - Dialects
- UnitOfWork - Session

💡 Svaki moderni framework koristi design patterne - prepoznaj ih i razumjet ćeš framework bolje!

🎯 Zaključak i Resursi

🎭 KLJUČNE PORUKE

- Design patterns su provjerena rješenja
- Koriste se u svakom većem frameworku
- Omogućuju zajednički rječnik
- Ključni za tech intervjue u FAANG

⚠️ ČESTE POGREŠKE

- Over-engineering - ne koristi pattern gdje nije potreban
- Pattern za pattern - prvo definiraj problem
- Ignoriranje SOLID principa

📖 PREPORUČENI RESURSI

📕 "Design Patterns" - Gang of Four
📗 "Head First Design Patterns"
🌐 refactoring.guru - vizualni primjeri
🎮 sourcemaking.com - interaktivno

🚀 SLJEDEĆI KORACI

1. Prepoznaj patterne u postojećem kodu
2. Implementiraj 1 pattern tjedno
3. Code review - traži refaktoring prilike

"Dobar programer zna što napisati. Odličan programer zna što obrisati." - GoF filozofija