Objektové modelování, AD7B36OMO

Návrhové vzory
Tvorba objektů

Jiří Šebek

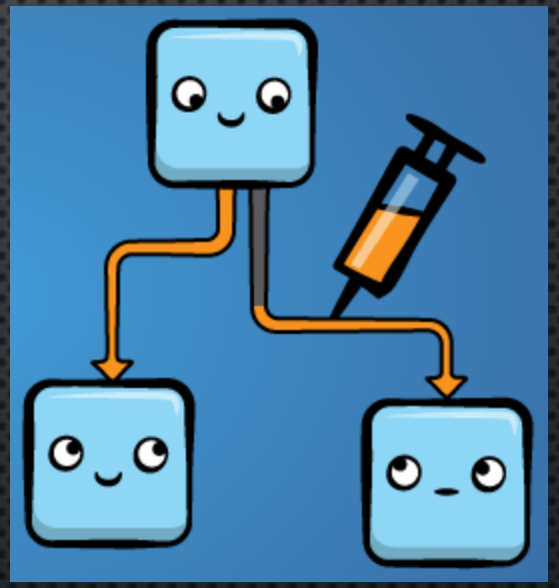
Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

https://edux.feld.cvut.cz/courses/AD7B36OMO

Přednáška je vytvořena na základě původních přednášek autorů: Karel Richta;Pavel Strnad, 2014

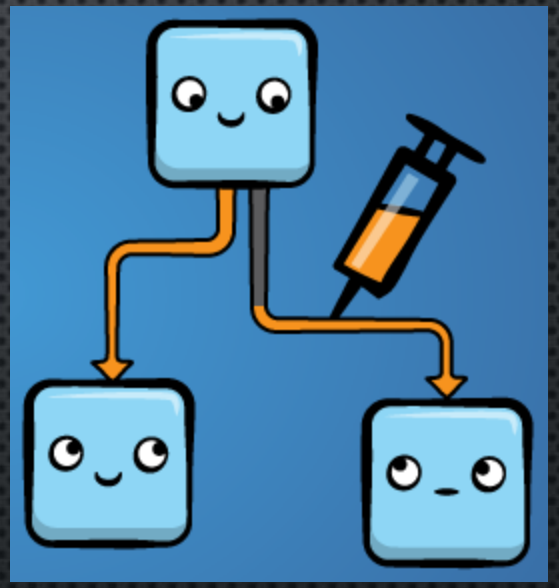
PRINCIPY IOC/DI V MODERNÍCH APLIKACÍCH

- Inversion of Control (IoC)
  - „obrácené řízení “
- Dependency Injection (DI)
  - „Vkládání závislostí“



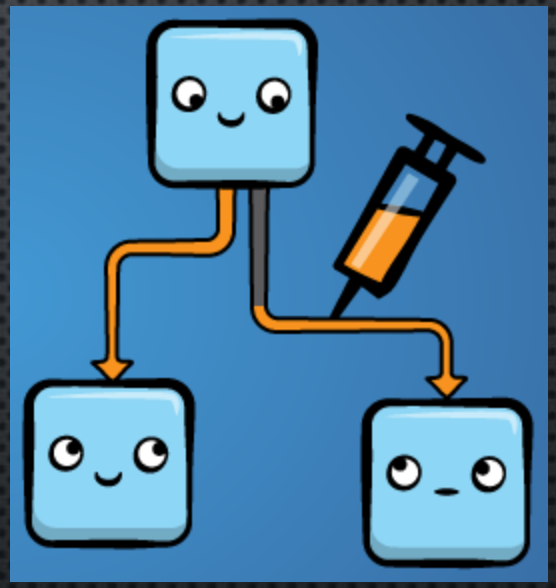
PRINCIPY IOC/DI V MODERNÍCH APLIKACÍCH

- **Inversion of Control (IoC)**
  - **„obrácené řízení “**
  - "IoC je, když máte pro vás někdo jiný vytvořit objekty." Takže namísto psaní "nového MyObject" ve vašem kódu je objekt vytvořen někým jiným. Tento "někdo jiný" se obvykle označuje jako kontejner s IOC.
  - IoC, protože vytváření objektu je obrácené. Není to programátor, ale někdo jiný, kdo vytváří objekt.
- Dependency Injection (DI)
  - „Vkládání závislostí“



PRINCIPY IOC/DI V MODERNÍCH APLIKACÍCH

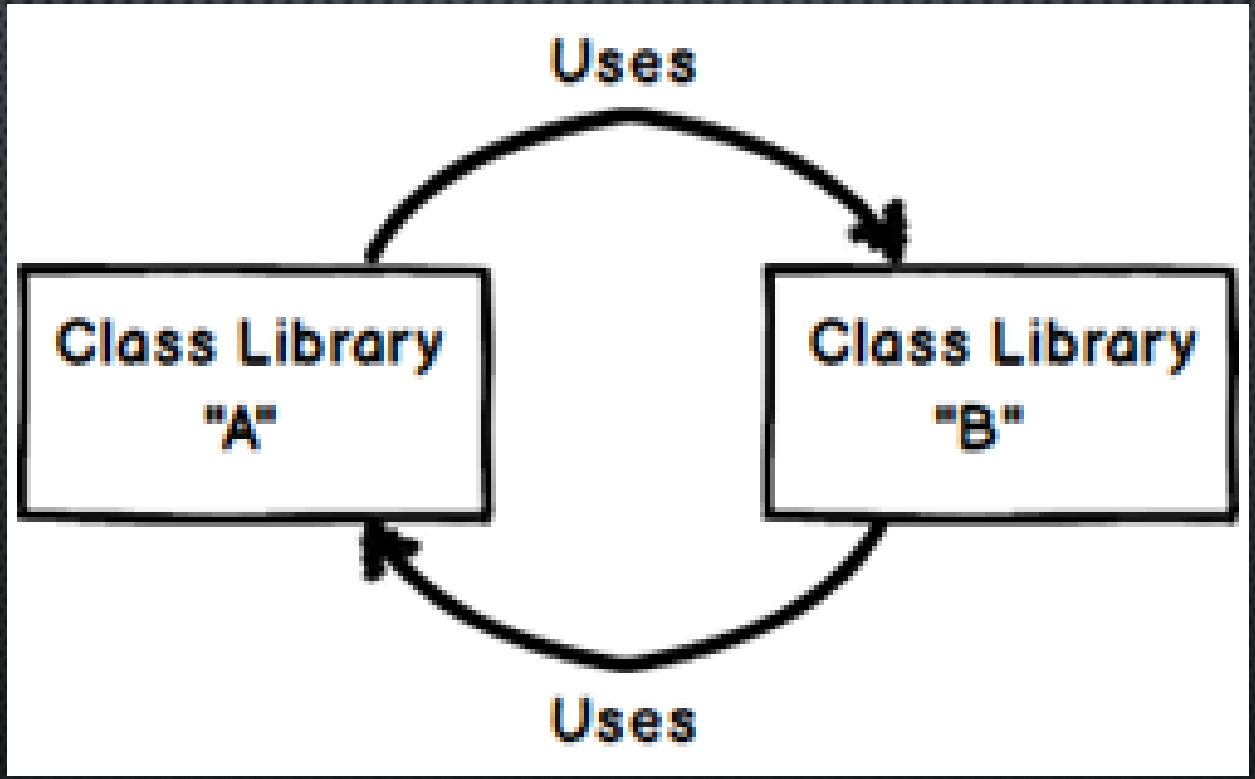
- **Inversion of Control (IoC)**
  - **„obrácené řízení "**
- Dependency Injection (DI)
  - „Vkládání závislostí"
  - Dependency Injection je zásadní pro umožnění správného testování.
  - Opak k hardcoded závislosti. (Ukážeme si) ☺



MOTIVACE

- Výhoda komponenty s přesně danou zodpovědností
- Lehká nahraditelnost
- Lehká **testovatelnost**
- Přehlednost
- Jednoduší struktura

CO JE ZÁVISLOST?



JAK PŘEDÁVAT?

- Nepředávat (není řešení ☺)
- Statika (špatně se mockuje, porušení bezpečnosti, hůř viditelné závislosti)
- Singleton
- Manualně (pracné)
- Service locator
- **IoC/DI ☺**

MOŽNÉ ZPŮSOBY INJEKTOVÁNÍ ZÁVISLOSTÍ

- Constructor injection – závislosti se nastavují parametry konstruktoru
- Setter (property) injection – závislost se injektuje setterem nebo přímo setováním public atributu
- Interface injection – třída implementuje rozhraní, které definuje injektované závislosti ve formě setterů

MOŽNÉ ZPŮSOBY INJEKTOVÁNÍ ZÁVISLOSTÍ

- CONSTRUCTOR

```
@Configuration
public class AppConfig {

    @Bean
    public Item item1() {
        return new ItemImpl1();
    }

    @Bean
    public Store store() {
        return new Store(item1());
    }
}
```

MOŽNÉ ZPŮSOBY INJEKTOVÁNÍ ZÁVISLOSTÍ

- SETTER

```
@Bean
public Store store() {
    Store store = new Store();
    store.setItem(item1());
    return store;
}
```

MOŽNÉ ZPŮSOBY INJEKTOVÁNÍ ZÁVISLOSTÍ

- FIELD

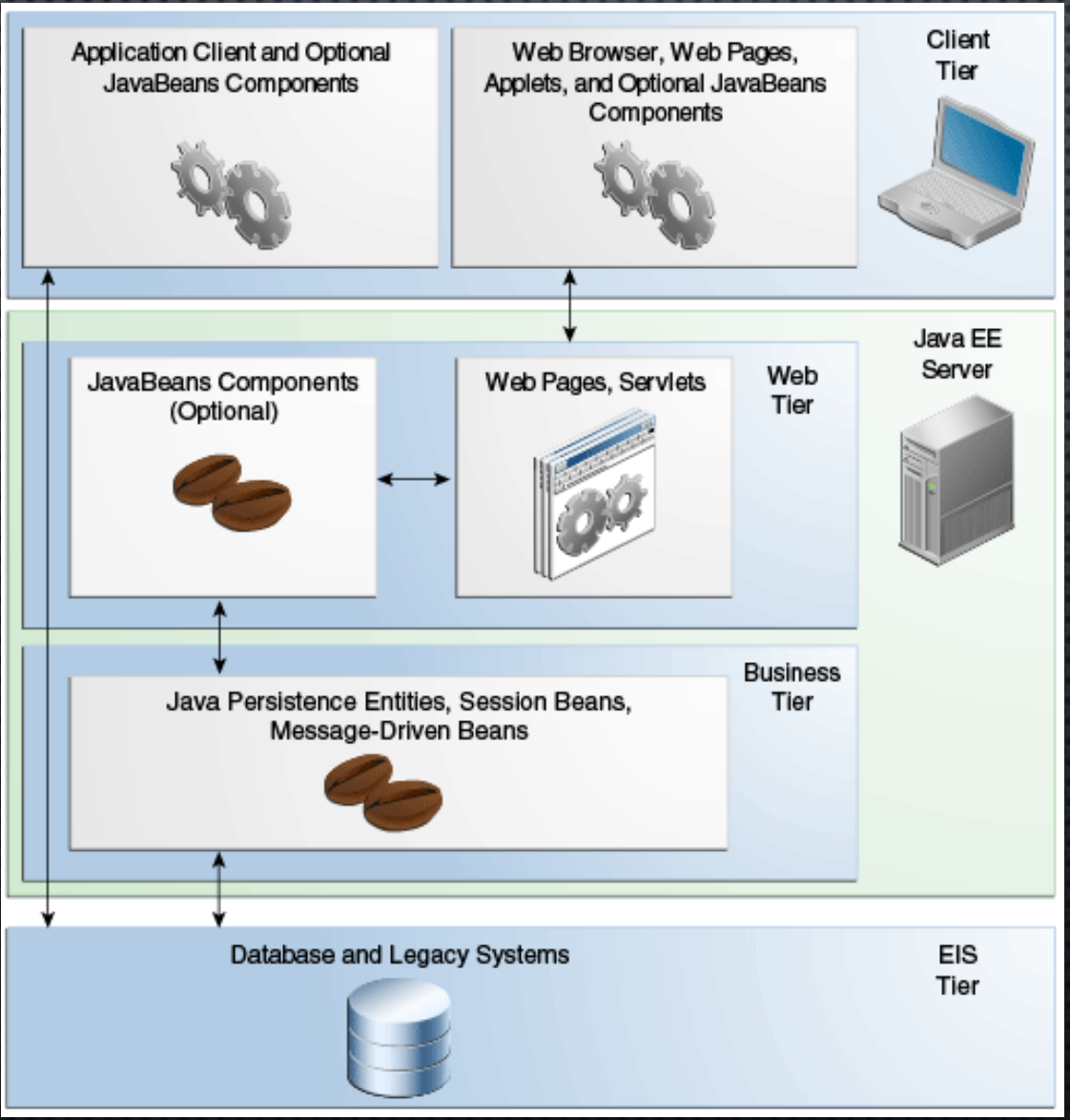
```
public class Store {
    @Autowired
    private Item item;
}
```

Existují čtyři režimy automatického přidávání komponent pomocí konfigurace XML:

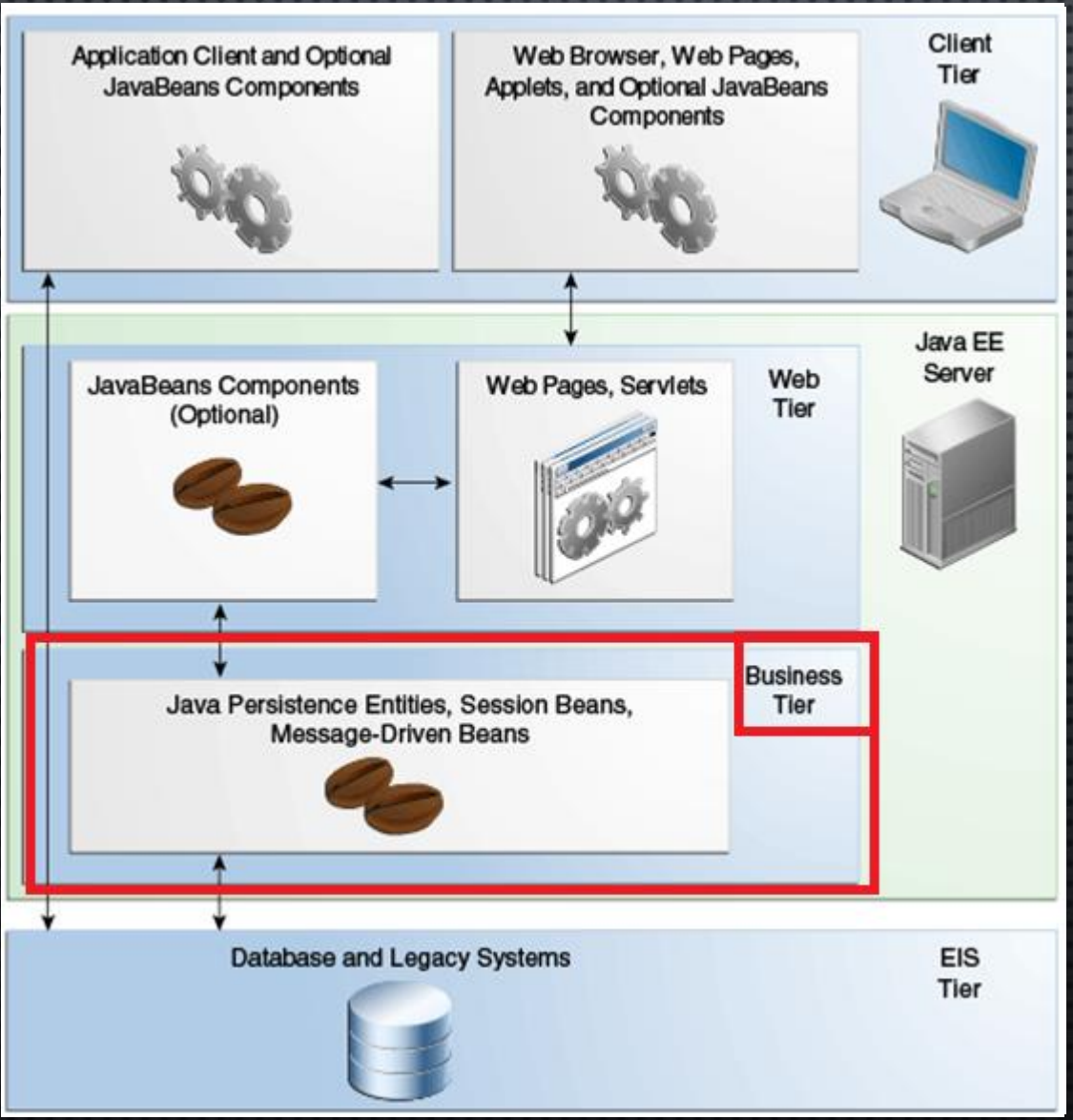
no:
byName:
byType:
constructor:

```
@Bean(autowire = Autowire.BY_TYPE)
public class Class1 {
```

KDE TO VŠECHNO POUŽIJEME?



KDE TO VŠECHNO POUŽIJEME?



SERVICE LAYER - PŘÍKLAD

- EJB ANOTACE

@COMPONENT

PRO SPRING

```
@Stateless
public class AddBookIntoLibraryImpl implements AddBookIntoLibrary {

    @Inject
    private BookDAOImpl bookDAO;

    @Inject
    private LibraryDAOImpl libraryDAO;

    @Inject
    private Logger logger;

    @Override
    public boolean addBook(long bookId, long libId) {
        Book book = bookDAO.getById(bookId);
        Library library = libraryDAO.getById(libId);

        List<Book> bookList = library.getBooks();
        if (bookList == null) bookList = new ArrayList<Book>();

        //if (book.getPublishingHouse() == null) book.setPublishingHouse(new ArrayList<Library>());
        //book.getKnihovnaKnihy().add(library);
        if (library.getBooks() == null) library.setBooks(new ArrayList<Book>());
        library.getBooks().add(book);

        try{
            bookDAO.save(book);
        }catch(Exception e){
            logger.log(Level.WARNING, "ERRROR: "+ book.getName()+" was not added - bookDAO.save failed");
            return false;
        }
        try{
```

CONTROLLER/ BUSSINES BEAN LAYER - PŘÍKLAD

POMOCÍ @INJECT NEBO @AUTOWIRED

```
@Model
public class AddBookController {


    @Inject
    private BookDAO bookDAO;

    @Inject
    private LibraryDAO libraryDAO;

    @Inject
    AddBookIntoLibrary addBookIntoLibrary;

    @Inject
    Logger logger;

    @Inject
    private FacesContext facesContext;

    @Produces
    @Named
    Book newBook;

    @Produces
    @Named
    Library newLibrary;

    @PostConstruct
    public void initEntity() {
        newBook = new Book();
        newLibrary = new Library();
    }

    public void create() {
```

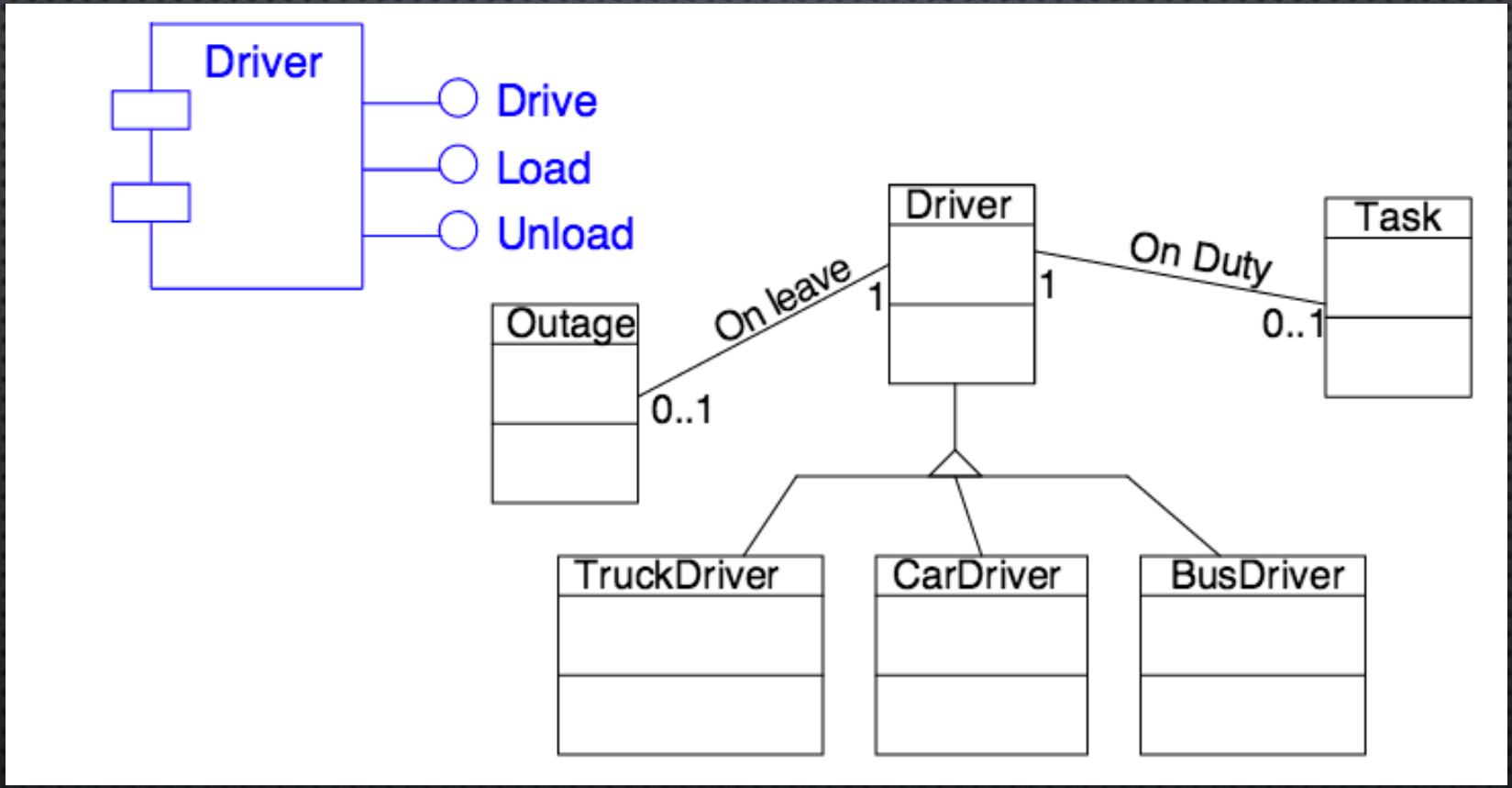
DALŠÍ VARIANTA

- Pro sestavení komponenty/beany/kontejneru můžeme použít konfiguraci XML (není potřeba poté anotace).
  Zde je jeden způsob, jak ručně vytvořit instanci kontejneru:

```
  <bean id="store" class="org.baeldung.store.Store">
    <constructor-arg type="ItemImpl1" index="0" name="item" ref="item1" />
</bean>
```

OBJEKTY VS KOMPONENTY

OBJEKTY VS KOMPONENTY



OBJECT-BASED DESIGN

– IDENTITY ORIENTED – DOMAIN ABSTRACTION

COMPONENT-BASED DESIGN

– SERVICE ORIENTED – FUNCTIONAL ABSTRACTION

KDO ROZUMÍ KOMPONENTÁM?

- INTERPRET!

APPLICATION SERVERS

- Webový kontejner (1)

Komponenta webového rozhraní a webový server
Komponentní server Servlet / JSF / JSP
Kontejner spravuje svůj životní cyklus, požadavek odesílání, poskytuje kontextové informace

- Klientský kontejner pro aplikace (2)

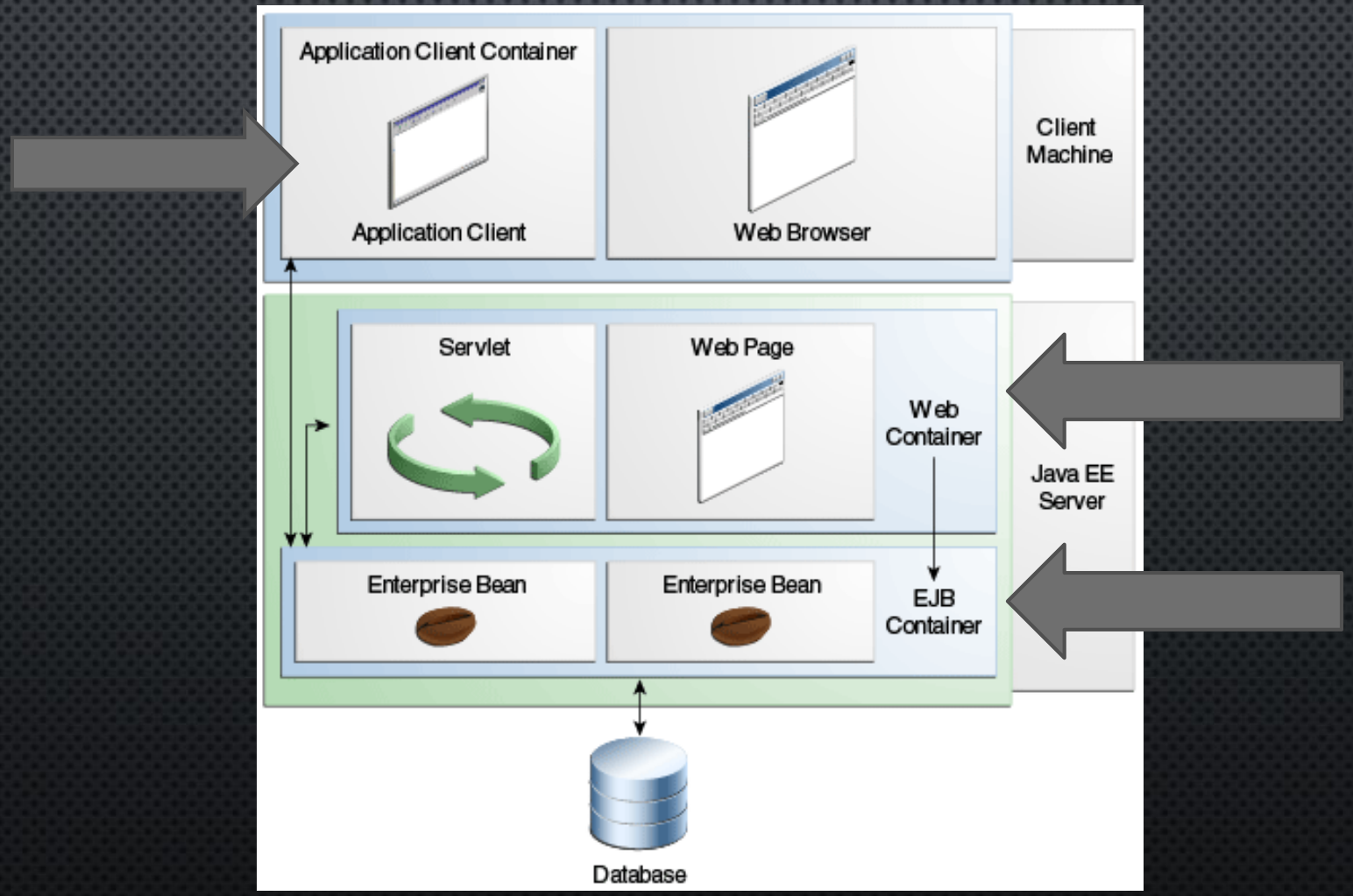
Klienti aplikací Java EE používající komponenty serveru Java EE

- EJB kontejner (3)

Rozhraní mezi EJB, která poskytuje bussiness logiku a server Java EE
EJB kontejner řídí provádění EJB

JAVA EE

APPLICATION SERVERS



SEZNAM FRAMEWORKŮ

Embedded frameworky

Google Guice
PicoContainer

Aplikační frameworky

JBoss Microcontainer
Spring framework

Standardní

Java EE EJB 2,3

JINÁ MOŽNOST NEŽ IOC/DI

- SERVICE LOCATOR

SERVICE LOCATOR

- VZOR LOKÁTORU SLUŽBY POSKYTUJE JINÝ PŘÍSTUP.
- JEDNÁ SE O JEDEN REGISTR PRO VŠECHNY SLUŽBY, KTERÉ JSOU POUŽÍVÁNY NAŠÍ APLIKACÍ

```
public class ServiceLocator {

    private static ServiceLocator locator;

    private Machine machine;

    private ServiceLocator() {
        Map<Key, Value> maps = new HashMap<Key, Value>();
        _________
        _________
        _________

        machine = new Machine(maps);
    }

    public static ServiceLocator getInstance() {
        if (locator == null) {
            locator = new ServiceLocator();
        }
        return locator;
    }

    public Machine machine() {
        return machine;
    }
}
```

KÓD NEBO KONFIGURAČNÍ SOUBOR?

Čitelnost?

Pouze hodnoty a ne logika aplikace?

DOTAZY?

```
@Path("/")
public class HelloWorld {
    @Inject
    HelloService helloService;

    @GET
    @Path("/json/{name}")
    @Produces("application/json")
    public String getHelloWorldJSON(@PathParam("name") String name) {
  return "{\"result\":\"" + helloService.createHelloMessage(name) + "\"}";
    }
```