Notatki do egzaminu – **Programowanie aplikacji internetowych**

Poniższe notatki zawierają **najważniejsze** zagadnienia zaznaczone pogrubioną czcionką. Są to **kluczowe** punkty, które warto zapamiętać, aby tworzyć kompletne aplikacje internetowe w technologii Java EE.

Wykład 1 – **Wprowadzenie do Java EE i serwera GlassFish**

1. **Czym jest Java EE?**
   - **Java Enterprise Edition** (**Java EE**) to platforma do tworzenia **skalowalnych** i **wielowarstwowych** aplikacji korporacyjnych w języku Java.
   - Zapewnia zestaw **specyfikacji (API)** i usług usprawniających tworzenie dużych projektów webowych.
2. **GlassFish – serwer aplikacji**
   - **GlassFish** to referencyjna implementacja **Java EE**.
   - Pozwala na **wdrażanie (deploy)** aplikacji Java EE.
   - Zarządza **kontenerem webowym** (serwlety, JSP), **kontenerem EJB**, oraz innymi usługami (np. obsługa baz danych, bezpieczeństwo).
3. **Czym jest IDE (Integrated Development Environment)?**
   - **Środowisko programistyczne** (np. NetBeans, Eclipse, IntelliJ), zawierające edytor kodu, narzędzia do debugowania oraz do zarządzania projektami.
   - **NetBeans IDE** – oficjalne środowisko wspierane przez Oracle, z wbudowaną obsługą **GlassFish**, co ułatwia tworzenie i uruchamianie aplikacji **Java EE**.

Wykład 2 – **Serwlety** w NetBeans

1. **Czym są serwlety?**
   - **Serwlety** to podstawowa technologia **Java EE** do obsługi **żądań HTTP** z przeglądarek.
   - Działają wewnątrz **kontenera webowego** (np. GlassFish), który zarządza ich cyklem życia (tworzenie, obsługa żądań, niszczenie).
2. **Tworzenie pierwszego serwletu w NetBeans**
   - **Krok 1**: Utwórz **nowy projekt** typu *Java Web → Web Application*.
   - **Krok 2**: Dodaj **nowy serwlet** (importy z `javax.servlet` i `javax.servlet.http`).
   - **Krok 3**: Zaimplementuj metody `doGet()` i/lub `doPost()`, przetwarzając dane żądania i generując odpowiedź.
3. **Podstawowe możliwości serwletów**
   - Odczyt parametrów z żądania: `request.getParameter("nazwaParametru")`.

- Wysyłanie odpowiedzi HTML: `response.getWriter().println("<html>...")`.
- Zarządzanie nagłówkami **HTTP** (np. `response.setHeader("Content-Type", "text/html")`).

Schemat wywołania serwletu:

```
flowchart LR
    A[Przeglądarka] --> B[Żądanie HTTP /serwlet]
    B --> C[Kontener Serwletu (GlassFish)]
    C --> D[Wywołanie metody doGet/doPost]
    D --> E[Przetwarzanie żądania w serwlecie]
    E --> F[Odpowiedź: strona HTML lub dane]
    F --> A
```

Wykład 3 – **Zaawansowane zagadnienia w serwletach**

1. **Obsługa sesji i ciasteczek**
   - **Sesja HTTP** (`HttpSession`) – przechowuje dane użytkownika pomiędzy żądaniami.
   - **Ciasteczka (Cookies)** – niewielkie informacje zapisywane w przeglądarce.
     - Ustawianie ciasteczka:

       ```
       Cookie cookie = new Cookie("nazwa", "wartosc");
       response.addCookie(cookie);
       ```

     - Odczyt ciasteczek:

       ```
       Cookie[] cookies = request.getCookies();
       ```

2. **Przekazanie sterowania innemu serwletowi**
   - Używając `RequestDispatcher.forward()` lub `RequestDispatcher.include()`, możemy przekierować żądanie do **innego serwletu** lub **JSP**.

3. **Filtry**
   - **Filtry** (ang. filters) „przechwytują” żądanie **przed** dotarciem do serwletu.
   - Służą m.in. do **autoryzacji**, **logowania** czy modyfikowania nagłówków.

Wykład 4 – **Technologia JavaServer Pages (JSP)**

1. **Czym jest JSP?**
   - **JSP** łączy **kod HTML** z **kodem Javy** po stronie serwera.
   - Kod JSP jest tłumaczony do serwletu i uruchamiany w **kontenerze webowym**.

2. **Tworzenie prostej aplikacji w JSP**
    - Utwórz plik `.jsp` w katalogu `Web Pages`.
    - Wstaw fragmenty **HTML** oraz **Java** (np. `<% %>`) w kodzie strony.
3. **Dodatkowe możliwości JSP**
    - **Dyrektywy** (np. `<%@ page import="java.util.*" %>`).
    - **Skrypty** (`<% int liczba = 10; %>`).
    - **Wyrażenia** (`<%= liczba %>`).
    - **JSTL i EL** (tematy w kolejnych wykładach).

Wykład 5 – **JavaBeans**, prosta aplikacja (HTML, serwlet, JSP) i **EL/JSTL**

1. **Komponenty JavaBeans**
    - **JavaBeans** to klasy z polami prywatnymi i metodami `get/set`.
    - Służą do **przechowywania** danych lub logiki biznesowej.
    - **Oddzielają** warstwę prezentacji (JSP) od logiki (serwlet, klasa).
2. **Przykładowa aplikacja**
    1. **Formularz HTML** →
    2. **Serwlet** odbiera dane →
    3. **Obiekt JavaBeans**, w którym zapisujemy dane →
    4. **Strona JSP**, która wyświetla dane z JavaBeans.
3. **Expression Language (EL)**
    - Składnia: `${nazwaPola}` lub `#{nazwaPola}`.
    - Ułatwia „**wstrzykiwanie**” wartości z obiektów (request, session, JavaBeans) w kodzie JSP/JSF.
4. **Java Standard Tag Library (JSTL)**
    - Zestaw znaczników do **pętli**, **warunków**, **formatowania** itp.
    - Przykład **pętli**:

```
<c:forEach var="item" items="${listaRzeczy}">
  ${item.name} <br/>
</c:forEach>
```

Wykład 6 – **Obsługa baz danych (JDBC)** i JSTL

1. **Połączenie z relacyjną bazą danych**
    - Często używany przykład to **Apache Derby** (wbudowana baza w NetBeans).
    - Tworzymy bazę, tabele, a następnie łączymy się z nimi w kodzie serwletu, używając **JDBC**.

2. **Podstawowe zapytania**
    - **SELECT**, **INSERT**, **UPDATE**, **DELETE**.
    - Przykład:

```
Connection conn = DriverManager.getConnection(
    "jdbc:derby://localhost:1527/baza",
    "uzytkownik",
    "haslo"
);
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery("SELECT * FROM Pracownicy");
```

3. **Obsługa bazy danych w JSTL**
    - JSTL posiada **tagi** `<sql:>` do zapytań w JSP.
    - Rzadziej stosowane w nowszych aplikacjach – częściej używa się **warstwy serwletów** lub **JPA**.

Wykład 7 – **Wprowadzenie do JavaServer Faces (JSF)**

1. **Framework JavaServer Faces**
    - **JSF** to framework MVC do tworzenia **interfejsu użytkownika** w aplikacjach **Java EE**.
    - Ma własne **znaczniki komponentów** (np. `<h:form>`, `<h:inputText>`), obsługę **walidacji** i konwersji.

2. **Managed Bean**
    - Klasa Javy oznaczona adnotacją `@ManagedBean` (lub w nowszych wersjach `@Named` + **CDI**).
    - Dostępna w plikach `.xhtml` przez **EL** (np. `#{bean.nazwaMetody}`).
    - Przechowuje **stan** i **logikę** związaną z widokiem.

3. **Walidacja danych**
    - **Atrybuty** typu `required="true"` czy `validatorMessage="Niepoprawne dane"`.
    - **Walidatory niestandardowe**: dedykowana klasa weryfikująca dane.
    - **Wyrażenia regularne**: `<f:validateRegex pattern="^[0-9]+$" />` (np. tylko cyfry).

Wykład 8 – **JSF z JPA (Java Persistence API)**

1. **Tworzenie encji JPA**
    - **Encja** to klasa odzwierciedlająca tabelę w bazie.
    - Oznaczamy adnotacją `@Entity`.
    - Przykład:

```java
@Entity
public class Uzytkownik {
  @Id
  private Long id;
  private String imie;
  private String nazwisko;
  // gettery i settery
}
```

2. **Kontroler JPA**
   - Klasa (fasada) zarządzająca **CRUD** (Create, Read, Update, Delete) za pomocą `EntityManager`.
   - **NetBeans** może generować taki kontroler automatycznie.
3. **Konfiguracja puli połączeń i źródła danych**
   - W **GlassFish** tworzymy **Connection Pool** i **Data Source** (JNDI).
   - W `persistence.xml` wskazujemy, z której puli połączeń korzystać.
4. **Dodawanie pól do istniejącej encji**
   - Po dodaniu nowych pól w klasie encji, konieczne jest **zaktualizowanie** bazy oraz **odświeżenie** modeli w NetBeans.
5. **Generowanie aplikacji JSF na podstawie encji**
   - **NetBeans** pozwala wygenerować gotowe widoki **CRUD** dla każdej encji.

Wykład 9 – **Projekt aplikacji Java Enterprise**: EJB i klient aplikacji

1. **Tworzenie projektu Java EE w NetBeans**
   - Możemy stworzyć projekt typu **Enterprise Application**, zawierający moduł **EJB** oraz moduł **Web**.
2. **Moduł EJB**
   - Tworzymy w nim **session bean** (np. `@Stateless` lub `@Stateful`).
   - Zawiera **metody biznesowe**, wywoływane przez klienta (web lub desktop).
3. **Projekt typu Java Class Library**
   - Jeśli chcemy rozdzielić logikę biznesową do osobnej biblioteki, tworzymy **Java Class Library** i dołączamy ją w projekcie.
4. **Dodawanie ziarna sesyjnego (session bean)**
   - `@Stateless public class MojeZiarno { ... }`.
   - Metody, np. `public String przywitaj(String imie) { ... }`.
5. **Dostęp do EJB (bean) z poziomu klienta**

- W aplikacji web (np. serwlet, JSF) można wstrzyknąć ziarno przez `@EJB`:

```
@EJB
private MojeZiarno ziarno;
```

Wykład 10 – **Usługa zegara (Timer Service)** i **CDI (Context and Dependency Injection)**

1. **Usługa zegara (Timer Service) w EJB**
   - Umożliwia **cykliczne** lub jednorazowe uruchamianie metod w session bean.
   - Przykład:

```
@Stateless
public class MojTimerBean {
  @Schedule(hour="*", minute="*/5", second="0", persistent=false)
  public void wykonywanaCo5Minut() {
    System.out.println("Metoda uruchamiana co 5 minut");
  }
}
```

2. **Context and Dependency Injection (CDI)**
   - Mechanizm **wstrzykiwania zależności** za pomocą adnotacji `@Inject`.
   - Ułatwia zarządzanie **cyklem życia** i konfiguracją obiektów.
   - **Integracja** z JSF – można wstrzyknąć bean do kontrolera i korzystać z niego w `.xhtml`.

3. **Przykład integracji CDI z JSF**
   - Klasa ziarna:

```
@Named
@RequestScoped
public class PrzykladoweZiarno {
  public String getPowitanie() {
    return "Hej ziarno!";
  }
}
```

   - W pliku `.xhtml`:

```
<h:outputText value="#{przykladoweZiarno.powitanie}" />
```

Podsumowanie – **Kluczowe elementy**:

- **GlassFish** – serwer aplikacji **Java EE**.
- **NetBeans** – IDE z wbudowaną integracją z **GlassFish**.
- **Serwlety** – obsługa żądań **HTTP**.
- **JSP** – generowanie stron **HTML** z wbudowanym kodem Javy.
- **JavaBeans** – klasy przenoszące **dane** i **logikę**.
- **Expression Language (EL)** / **JSTL** – ułatwienie **warstwy prezentacji**.
- **JDBC** / **JPA** – dostęp do **baz danych** (od „surowego" do obiektowego).
- **JSF** – framework **MVC** do budowy interfejsów użytkownika.
- **EJB** – **logika biznesowa** (session beans, timer service).
- **CDI** – **wstrzykiwanie zależności** w aplikacjach **Java EE**.