12. UTWORZENIE PROJEKTU I IMPLEMENTACJA

Aplikacja webowa będzie wykorzystywać usługę typu REST zaimplementowaną w ramach poprzedniego ćwiczenia.

12.1. Stwórz gradle'owy projekt aplikacji webowej np. *project-web-app*. Edytuj plik *build.gradle* i zmodyfikuj jego elementy, aby odpowiadał poniższej zawartości. Następnie kliknij prawym przyciskiem myszki na głównej ikonce projektu i wybierz *Gradle -> Refresh Gradle Project*.

```
plugins {
	id 'java'
	id 'org.springframework.boot' version '2.2.6.RELEASE'
	id 'io.spring.dependency-management' version '1.0.9.RELEASE'
}

group = 'com.project'
version = '1.0'

java {
    sourceCompatibility = JavaVersion.VERSION_1_8
    targetCompatibility = JavaVersion.VERSION_1_8
}

compileJava {                           // potrzebne, gdy edytor kodu źródłowego środowiska
	options.encoding = 'windows-1250' // Eclipse ma ustawione windowsowe kodowanie
}

compileTestJava{
	options.encoding = 'windows-1250'
}

configurations {
	developmentOnly
	runtimeClasspath {
		extendsFrom developmentOnly
	}
}

repositories {
   mavenCentral()
}

dependencies {
	developmentOnly 'org.springframework.boot:spring-boot-devtools'
	implementation 'org.springframework.boot:spring-boot-starter'
	implementation 'org.springframework.boot:spring-boot-starter-thymeleaf'
	implementation 'org.springframework.boot:spring-boot-starter-web'
	implementation 'org.springframework.data:spring-data-commons'
	implementation 'com.fasterxml.jackson.datatype:jackson-datatype-jsr310'
	testImplementation('org.springframework.boot:spring-boot-starter-test') {
		exclude group: 'org.junit.vintage', module: 'junit-vintage-engine'
	}
}

test {
   useJUnitPlatform() //aktywacja natywnego wsparcia JUnit 5 (od wersji 4.6 Gradle'a)
    testLogging {
        showStandardStreams = true //ustawia drukowanie komunikatów w konsoli
     }
}
```

12.2. Przekopiuj pakiet *com.project.model* utworzony w ramach poprzedniego ćwiczenia. W każdej klasie modelu kasujemy wszystkie adnotacje z *javax.persistence.** oraz *org.hibernate.annotations.**, a także pozostałe po ich usunięciu zbędne importy (nie trzeba usuwać poszczególnych importów, można użyć skrótu *CTRL + SHIFT + O*, który uporządkuje całą sekcję importów). Następnie w każdej klasie modelu, przed jej nazwą, dodajemy nową adnotację *@JsonIgnoreProperties*, dzięki której podczas tworzenia obiektów ignorowane będą właściwości komunikatów JSON-a niewchodzące w skład mapowanych klas. Poza tym można dodać adnotacje *@DateTimeFormat*, określające format prezentowanych dat i czasu.

```
package com.project.model;
...
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import org.springframework.format.annotation.DateTimeFormat;

@JsonIgnoreProperties(ignoreUnknown = true)
public class Projekt {
...
	@DateTimeFormat(pattern = "yyyy-MM-dd HH:mm:ss.SSS")
	private LocalDateTime dataCzasUtworzenia;

	@DateTimeFormat(pattern = "yyyy-MM-dd HH:mm:ss.SSS")
	private LocalDateTime dataCzasModyfikacji;

	@DateTimeFormat(pattern = "yyyy-MM-dd")
	private LocalDate dataOddania;

}
```

12.3. Dodaj pakiet *com.project.service* i przekopiuj do niego wszystkie interfejsy serwisów utworzone w ramach poprzedniego ćwiczenia (z pakietu o takiej samej nazwie co nowo utworzony, patrz realizacja usługi typu REST). Następnie dodaj nowe klasy je implementujące tj. *ProjektServiceImpl*, *ZadanieServiceImpl* i *StudentServiceImpl*, za pomocą środowiska Eclipse wygeneruj szkielety metod wymagających implementacji (patrz p. 4.7).

12.4. Dodajemy pakiet *com.project.config* i tworzymy w nim klasę *SecurityConfig* z poniższą zawartością. Jej zadaniem jest utworzenie obiektu klasy *RestTemplate* (jest to specjalna springowa klasa przeznaczona do komunikacji z REST API) oraz ustawienie loginu i hasła wykorzystywanego w uwierzytelnianiu typu *Basic Authentication*, zastosowanym w zaimplementowanej w poprzednim ćwiczeniu usłudze. Utworzony obiekt będzie mógł być wstrzykiwany w dowolnej klasie naszego projektu np. poprzez konstruktor.

```
package com.project.config;

import org.springframework.boot.web.client.RestTemplateBuilder;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.client.RestTemplate;

@Configuration
public class SecurityConfig {
		// dzięki adnotacji @Bean Spring uruchomi metodę i zarejestruje w kontenerze obiekt przez nią
  @Bean  // zwrócony, natomiast adnotacja @Autowired użyta w innej klasie spowoduje jego wstrzyknięcie
  RestTemplate customRestTemplate(RestTemplateBuilder restTemplateBuilder) {

      return restTemplateBuilder.basicAuthentication("admin", "admin").build();

  }
   // login i hasło można przechowywać w pliku np. application.properties
```

12.5. W kataloru *project-web-app\src\main\resources* utwórz plik tekstowy *application.properties*. Następnie edytuj go i zdefiniuj port serwera naszej aplikacji webowej oraz dodaj parametr z adresem serwera usługi REST tj.

```
server.port=8081
rest.server.url=http://localhost:8080
```

Spring uwzględnia wartość parametru *server.port* i uruchamia aplikację na odpowiednim porcie. Poza tym w każdej klasie można pobierać wartości z tego pliku konfiguracyjnego za pomocą adnotacji *@Value* (z pakietu *org.springframework.beans.factory.annotation.**), w której wskazuje się odpowiednią nazwę parametru np.

```
   @Value("${rest.server.url}")
   private String serverUrl;
```

12.6. W pakiecie *com.project.service* utwórz klasę pomocniczą *RestResponsePage*, będącą uniwersalnym szablonem służącym przekształcaniu komunikatów JSON-owych o specjalnej strukturze stosowanej przy stronicowaniu danych (przykład w punkcie 10.6). Zwracane dane modelu są konwertowane na obiekty i umieszczane w liście *content*, natomiast pozostałe zmienne przechowują parametry stronicowania i sortowania.

```
package com.project.service;

import java.util.ArrayList;
import java.util.List;
```

```
import org.springframework.data.domain.PageImpl;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonProperty;
import com.fasterxml.jackson.databind.JsonNode;

@JsonIgnoreProperties(ignoreUnknown = true)
public class RestResponsePage<T> extends PageImpl<T> {

   private static final long serialVersionUID = 1L;

   @JsonCreator(mode = JsonCreator.Mode.PROPERTIES)
   public RestResponsePage(@JsonProperty("content") List<T> content,
                           @JsonProperty("number") int number,
                           @JsonProperty("size") int size,
                           @JsonProperty("totalElements") Long totalElements,
                           @JsonProperty("pageable") JsonNode pageable,
                           @JsonProperty("last") boolean last,
                           @JsonProperty("totalPages") int totalPages,
                           @JsonProperty("sort") JsonNode sort,
                           @JsonProperty("first") boolean first,
                           @JsonProperty("numberOfElements") int numberOfElements) {

       super(content, PageRequest.of(number, size), totalElements);
    }

    public RestResponsePage(List<T> content, Pageable pageable, long total) {
       super(content, pageable, total);
    }

    public RestResponsePage(List<T> content) {
       super(content);
    }

    public RestResponsePage() {
       super(new ArrayList<>());
    }
}
```

12.7. W pakiecie *com.project.service* umieścimy jeszcze jedną klasę pomocniczą – *ServiceUtil*, z metodą generyczną *getPage* wysyłającą żądania typu GET i zwracająca obiekt powyższej klasy *RestResponsePage* (utworzony na podstawie JSON-owej odpowiedzi serwera). Pozostałe metody pomocnicze są wykorzystywane przy budowaniu adresów do zasobów z zadanymi parametrami stronicowania, sortowania i wyszukiwania.

```
import java.net.URI;

import org.springframework.core.ParameterizedTypeReference;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.http.HttpMethod;
import org.springframework.http.ResponseEntity;
import org.springframework.web.client.RestTemplate;
import org.springframework.web.util.UriComponentsBuilder;

public class ServiceUtil {

   public static <T> RestResponsePage<T> getPage(URI uri, RestTemplate restTemplate,
                                          ParameterizedTypeReference<RestResponsePage<T>> responseType) {
      ResponseEntity<RestResponsePage<T>> result = restTemplate.exchange(uri, HttpMethod.GET, null,
                                                                         responseType);
      return result.getBody();
   }

   public static URI getURI(String serverUrl, String resourcePath, Pageable pageable) {
      return getUriComponent(serverUrl, resourcePath)
            .queryParam("page", pageable.getPageNumber())
            .queryParam("size", pageable.getPageSize())
            .queryParam("sort", ServiceUtil.getSortParams(pageable.getSort())).build().toUri();
   }
```

```
   public static UriComponentsBuilder getUriComponent(String serverUrl, String resourcePath, Pageable
                                                                                                pageable) {
      return getUriComponent(serverUrl, resourcePath)
            .queryParam("page", pageable.getPageNumber())
            .queryParam("size", pageable.getPageSize())
            .queryParam("sort", ServiceUtil.getSortParams(pageable.getSort()));
   }

   public static UriComponentsBuilder getUriComponent(String serverUrl, String resourcePath) {
      return UriComponentsBuilder.fromUriString(serverUrl).path(resourcePath);
   }

   public static String getSortParams(Sort sort) {
      StringBuilder builder = new StringBuilder();
      if (sort != null) {
         String sep = "";
         for (Sort.Order order : sort) {
            builder.append(sep).append(order.getProperty()).append(",").append(order.getDirection());
            sep = "&sort=";
         }
      }
      return builder.toString();
   }
}
```

12.8. Przykładowa klasa serwisowa korzystająca z usługi REST. Zaimplementuj pozostałe klasy – *ZadanieServiceImpl* i *StudentServiceImpl*.

```
package com.project.service;
import java.net.URI;
import java.util.Optional;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.http.HttpEntity;
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;
import com.project.model.Projekt;

@Service
public class ProjektServiceImpl implements ProjektService {
   private static final Logger logger = LoggerFactory.getLogger(ProjektServiceImpl.class);

   @Value("${rest.server.url}") // adres serwera jest wstrzykiwany przez Springa, a jego wartość
   private String serverUrl;    // przechowywana w pliku src/main/resources/application.properties

   private final static String RESOURCE_PATH = "/api/projekty";

                                          // obiekt wstrzykiwany poprzez konstruktor, dzięki adnotacjom
   private RestTemplate restTemplate;      // @Configuration i @Bean zawartym w klasie SecurityConfig
                                           // Spring utworzy wcześniej obiekt, a adnotacja @Autowired
                                          // tej klasy wskaże element docelowy wstrzykiwania
                                          // (adnotacja może być pomijana jeżeli w klasie jest
                                          // tylko jeden konstruktor)
   @Autowired
   public ProjektServiceImpl(RestTemplate restTemplate) {
      this.restTemplate = restTemplate;
   }

   @Override
   public Optional<Projekt> getProjekt(Integer projektId) {
      URI url = ServiceUtil.getUriComponent(serverUrl, getResourcePath(projektId))
            .build()
            .toUri();
      logger.info("REQUEST -> GET {}", url);
      return Optional.ofNullable(restTemplate.getForObject(url, Projekt.class));
   }
```

```
    @Override
    public Projekt setProjekt(Projekt projekt) {
        if (projekt.getProjektId() != null) { // modyfikacja istniejącego projektu
            String url = getUriStringComponent(projekt.getProjektId());
            logger.info("REQUEST -> PUT {}", url);
            restTemplate.put(url, projekt);
            return projekt;
        } else {//utworzenie nowego projektu
            // po dodaniu projektu zwracany jest w nagłówku Location - link do utworzonego zasobu
            HttpEntity<Projekt> request = new HttpEntity<>(projekt);
            String url = getUriStringComponent();
            logger.info("REQUEST -> POST {}", url);
            URI location = restTemplate.postForLocation(url, request);
            logger.info("REQUEST (location) -> GET {}", location);
            return restTemplate.getForObject(location, Projekt.class);
            // jeżeli usługa miałaby zwracać utworzony obiekt a nie link to trzeba by użyć
            // return restTemplate.postForObject(url, projekt, Projekt.class);
        }
    }

    @Override
    public void deleteProjekt(Integer projektId) {
        URI url = ServiceUtil.getUriComponent(serverUrl, getResourcePath(projektId))
                .build()
                .toUri();
        logger.info("REQUEST -> DELETE {}", url);
        restTemplate.delete(url);
    }

    @Override
    public Page<Projekt> getProjekty(Pageable pageable) {
        URI url = ServiceUtil.getURI(serverUrl, getResourcePath(), pageable);
        logger.info("REQUEST -> GET {}", url);
        return getPage(url, restTemplate);
    }

    @Override
    public Page<Projekt> searchByNazwa(String nazwa, Pageable pageable) {
         URI url = ServiceUtil.getUriComponent(serverUrl, getResourcePath(), pageable)
                 .queryParam("nazwisko", nazwa)
                 .build().toUri();
         logger.info("REQUEST -> GET {}", url);
         return getPage(url, restTemplate);
    }

    // metody pomocnicze
    private Page<Projekt> getPage(URI uri, RestTemplate restTemplate) {
        return ServiceUtil.getPage(uri, restTemplate,
                                           new ParameterizedTypeReference<RestResponsePage<Projekt>>() {});
    }

    private String getResourcePath() {
        return RESOURCE_PATH;
    }

    private String getResourcePath(Integer id) {
        return RESOURCE_PATH + "/" + id;
    }

    private String getUriStringComponent() {
        return serverUrl + getResourcePath();
    }

    private String getUriStringComponent(Integer id) {
        return serverUrl + getResourcePath(id);
    }
}
```

12.9. Przykładowy kontroler aplikacji webowej.

```
package com.project.controller;

import javax.validation.Valid;
import org.springframework.data.domain.Pageable;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.client.HttpStatusCodeException;
import com.project.model.Projekt;
import com.project.service.ProjektService;

@Controller
public class ProjectController {
   private ProjektService projektService;

   //@Autowired – przy jednym konstruktorze wstrzykiwanie jest zadaniem domyślnym, adnotacja nie jest potrzebna
   public ProjectController(ProjektService projektService) {
      this.projektService = projektService;
   }
                              // metodę wywołamy wpisując w przeglądarce np. http://localhost:8081/projektList lub
   @GetMapping("/projektList") // http://localhost:8081/projektList?page=0&size=10&sort=dataCzasModyfikacji,desc
   public String projektList(Model model, Pageable pageable) {

                               // za pomocą zmiennej model i metody addAttribute przekazywane są do widoku obiekty
                               // zawierające dane projektów
      model.addAttribute("projekty", projektService.getProjekty(pageable).getContent());
      return "projektList";    // metoda zwraca nazwę logiczną widoku, Spring dopasuje nazwę do odpowiedniego
   }                           // szablonu tj. project-web-app\src\main\resources\templates\projektList.html

   @GetMapping("/projektEdit")
   public String projektEdit(@RequestParam(required = false) Integer projektId, Model model) {
      if(projektId != null) {
         model.addAttribute("projekt", projektService.getProjekt(projektId).get());
      }else {
         Projekt projekt = new Projekt();
         model.addAttribute("projekt", projekt);
      }
      return "projektEdit"; // metoda zwraca nazwę logiczną widoku, Spring dopasuje nazwę do odpowiedniego
   }                        // szablonu tj. project-web-app\src\main\resources\templates\projektEdit.html

   @PostMapping(path = "/projektEdit")
   public String projektEditSave(@ModelAttribute @Valid Projekt projekt, BindingResult bindingResult) {
                         // parametr BindingResult powinien wystąpić zaraz za parametrem opatrzonym adnotacją @Valid
      if (bindingResult.hasErrors()) {
         return "projektEdit"; // wracamy do okna edycji, jeżeli przesłane dane formularza zawierają błędy
      }
      try {
         projekt = projektService.setProjekt(projekt);
      } catch (HttpStatusCodeException e) {
         bindingResult.rejectValue(null, String.valueOf(e.getStatusCode().value()),
                                                                    e.getStatusCode().getReasonPhrase());
         return "projektEdit";
      }
      return "redirect:/projektList"; // przekierowanie do listy projektów, po utworzeniu lub modyfikacji projektu
   }

   @PostMapping(params="cancel", path = "/projektEdit") // metoda zostanie wywołana, jeżeli przesłane
   public String projektEditCancel() {                  // żądanie będzie zawierało parametr 'cancel'
      return "redirect:/projektList";
   }

   @PostMapping(params="delete", path = "/projektEdit")             // metoda zostanie wywołana, jeżeli przesłane
   public String projektEditDelete(@ModelAttribute Projekt projekt) {// żądanie będzie zawierało parametr 'delete'
      projektService.deleteProjekt(projekt.getProjektId());
      return "redirect:/projektList";
   }
}
```

12.10. Przykładowy szablon (*Thymeleaf*) ekranu z listą projektów (plik *project-web-app\src\main\resources\templates\projektList.html*).

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<link th:href="@{/css/list-style.css}" rel="stylesheet" /> <!-- plik z ...\resources\static\css\list-style.css -->
<title>Lista projektów</title>
</head>
<body>
  <div class="root">
    <h1>Lista projektów</h1>
      <p><a th:href="@{/projektEdit}">Dodaj projekt</a></p>
      <table>
        <thead>
          <tr>
            <th>Id</th>
            <th>Nazwa</th>
            <th>Opis</th>
            <th>Utworzony</th>
            <th>Zmodyfikowany</th>
            <th>Data oddania</th>
            <th>Edit</th>
          </tr>
        </thead>
        <tbody>
          <tr th:each="projekt : ${projekty}">
            <td th:text="${projekt.projektId}">Id</td>
            <td th:text="${projekt.nazwa}">Nazwa</td>
            <td th:text="${projekt.opis}">Opis</td>
            <td th:text="${#temporals.format(projekt.dataCzasUtworzenia, 'yyyy-MM-dd HH:mm:ss')}">Utworzony</td>
            <td th:text="${#temporals.format(projekt.dataCzasModyfikacji, 'yyyy-MM-dd HH:mm:ss')}">Zmodyfikowany</td>
            <td th:text="${#temporals.format(projekt.dataOddania, 'yyyy-MM-dd')}">Data oddania</td>
            <td>
              <a th:href="@{/projektEdit(projektId=${projekt.projektId})}">Edytuj</a><br>
              <a th:href="@{/projektEdit(projektId=${projekt.projektId},delete='true')}">Usuń</a>
            </td>
          </tr>
        </tbody>
      </table>
  </div>
</body>
</html>
```

12.11. Przykładowy szablon (*Thymeleaf*) ekranu edycji projektu (plik *project-web-app\src\main\resources\templates\projektEdit.html*).

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<link th:href="@{/css/edit-style.css}" rel="stylesheet" />    <!-- plik z ...\resources\static\css\edit-style.css -->
<title>Edycja projektu</title>
</head>
<body>
  <div class="root" th:with="isDelete=${#strings.equalsIgnoreCase(param.delete,'true')}">
    <form action="#" th:action="@{/projektEdit}" th:object="${projekt}" method="POST"
      th:with="akcja=*{projektId} ? (${isDelete}?'delete':'update') :'create', opis=*{projektId} ?
                                                  (${isDelete}?'Usuń':'Aktualizuj') : 'Utwórz'" autocomplete="off">

      <h1 th:text="${opis} + ' projekt'">Edytuj projekt</h1>

      <div class="err" th:if="${#fields.hasErrors('*')}">
        BŁĘDY:
        <ul>
          <li th:each="err : ${#fields.errors('*')}" th:text="${err}">Wprowadzone dane są niepoprawne!</li>
        </ul>
      </div>

      <div class="container">
        <div class="btns-panel">
          <input class="btn" type="submit" name="create" value="create" th:name="${akcja}" th:value="${opis}" />
          <input class="btn" type="submit" name="cancel" value="Anuluj" />
        </div>
        <div th:if="*{projektId}">
          <label for="projektId" class="lbl">Id:</label>
          <input th:field="*{projektId}" class="fld" readonly />
        </div>
```

```
        <div>
          <label for="nazwa" class="lbl">Nazwa:</label>
          <input th:field="*{nazwa}" class="fld" th:class="${#fields.hasErrors('opis')}? 'err' : 'fld'" size="45" />
          <span class="err" th:if="${#fields.hasErrors('nazwa')}" th:errors="*{nazwa}">Error</span>
        </div>
        <div>
          <label for="opis" class="lbl">Opis:</label>
          <textarea class="fld" rows="3" cols="47" th:field="*{opis}">Opis</textarea>
        </div>
        <div>
          <label for="dataOddania" class="lbl">Data oddania:</label>
          <input th:field="*{dataOddania}" class="fld" type="text" size="10" /><i>(RRRR-MM-DD)</i>
        </div>
        <div th:if="*{dataCzasUtworzenia}">
          <label for="dataCzasUtworzenia" class="lbl">Utworzony:</label>
          <input th:field="*{dataCzasUtworzenia}" class="fld" type="text" size="23" readonly />
        </div>
        <div th:if="*{dataCzasModyfikacji}">
          <label for="dataCzasModyfikacji" class="lbl">Zmodyfikowany:</label>
          <input th:field="*{dataCzasModyfikacji}" class="fld" type="text" size="23" readonly />
        </div>
      </div>
    </form>
  </div>
</body>
</html>
```

12.12. Zaimplementuj pełną funkcjonalność systemu pozwalającą dodawać, modyfikować i usuwać dane projektów, zadań i studentów. Opcjonalnie, tylko dla chętnych realizacja mechanizmów stronicowania i wyszukiwania danych projektów oraz studentów, a także zabezpieczenie aplikacji przed niepożądanym dostępem. Pamiętaj o uruchomieniu usługi REST przed włączeniem aplikacji webowej. Lista projektów będzie dostępna pod adresem http://localhost:8081/projektList.

PRZYDATNE SKRÓTY

CTRL + SHIFT + L – pokazuje wszystkie dostępne skróty

CTRL + SHIFT + F – formatowanie kodu

SHIFT + ALT + R – zmiana nazwy klasy, metody lub zmiennej itp., trzeba wcześniej ustawić kursor na nazwie

SHIFT + ALT + L – utworzenie zmiennej z zaznaczonego fragmentu kodu

SHIFT + ALT + M – utworzenie metody z zaznaczonego fragmentu kodu

CTRL + ALT + *STRZAŁKA W GÓRĘ* – skopiowanie linijki i wklejenie w bieżącym wierszu

CTRL + ALT + *STRZAŁKA W DÓŁ* – skopiowania bieżącej linijki i wklejenie poniżej

CTRL + SHIFT + O – automatyczne dodawanie i porządkowanie sekcji importów

CTRL + 1 – „zrób to co chcę zrobić”, m.in. sugestie rozwiązań bieżącego problemu

CTRL + Q – przejście do miejsca ostatniej modyfikacji

F11 – debugowanie aplikacji

CTRL + F11 – uruchomienie aplikacji

CTRL + M – powiększenie/zmniejszenie widoku w perspektywie

Ustawienie kursora np. na wywołaniu metody, typie zmiennej, klasie importu itp. i wciśnięcie **F3** powoduje przejście do kodu źródłowego wywoływanej metody, klasy zmiennej, klasy importu itd.