

# Java - zadania praktyczne

## *Toruńska Agencja Rozwoju Regionalnego*

Software Development Academy

## Zadania praktyczne z Javy

- Pobierz projekt mavenowy z <https://bitbucket.org/sdatarr/exam/> lub wykonaj w konsoli polecenie:

```
git clone https://repo-sda@bitbucket.org/sdatarr/exam.git
```

- Zaimportuj projekt **exam** do IDE (*IntelliJ IDEA*)
- Upewnij się, że zaciągnęły się wszystkie zależności; w razie potrzeby w IDE wybierz z menu **Build** > **Build Project** lub z bocznego menu **Maven** > **exam** > **Lifecycle** > **package**

- W projekcie znajdują się dwa zadania do wykonania:

1. Zadanie dotyczące narysowania prostego wzorca
2. Aplikacja REST do rozwinięcia

Nazwij plik swoim imieniem i nazwiskiem!



• **Przed zakończeniem egzaminu**, katalog z projektem należy spakować (np. ZIPem) i wysłać mailem do **adam93@post.pl** **mikruczkowski@gmail.com** i **a.ciesla@sdacademy.pl**

- Przed spakowaniem należy usunąć katalog **target** z projektu; ręcznie lub z bocznego menu wybrać **Maven** > **exam** > **Lifecycle** > **clean**
- Podczas rozwiązywania zadań **można** - w razie potrzeby - korzystać z *przeglądarki internetowej* w celu przeszukiwania zasobów WWW.
- **Nie wolno** natomiast używać jakiegokolwiek komunikacji przez internet.
- Czas trwania **180 minut**.  
**POWODZENIA!**

# Zadanie 1. Kwadrat z przekątnymi

W pakiecie `pl.sdacademy.tarr.exam.ex1` znajduje się klasa `Pattern` posiadająca metodę `public String prepareSquareWithDiagonals(char sign, int size)`, gdzie:

- `char sign` - to znak, którym wypełniamy krawędzie i przekątną kwadratu
- `int size` - to liczba tych znaków (rozdzielonych spacją!) w krawędzi kwadratu

Zaimplementuj metodę `prepareSquareWithDiagonals`, która zwróci w zmiennej typu `String` narysowany kwadrat wraz z przekątnymi, zgodnie z podanym rozmiarem i znakiem do rysowania.

Pomocne mogą być testy jednostkowe, które znajdują się w katalogu `src/test/java`.

Poniżej przykłady kwadratów dla różnych znaków i rozmiarów.

Przykład dla `sign = '#'` i `size = 1`

```
#
```

Przykład dla `sign = 'x'` i `size = 2`

```
x x
x x
```

Przykład dla `sign = '@'` i `size = 5`

```
@ @ @ @ @
@ @   @ @
@   @   @
@ @   @ @
@ @ @ @ @
```

Przykład dla `sign = '*'` i `size = 7`

```
* * *      * * * *
* *        * *
*   *   *   *
*         *   *
*   *   *   *
* *        * *
* * *      * * * *
```

## Zadanie 2. Aplikacja REST

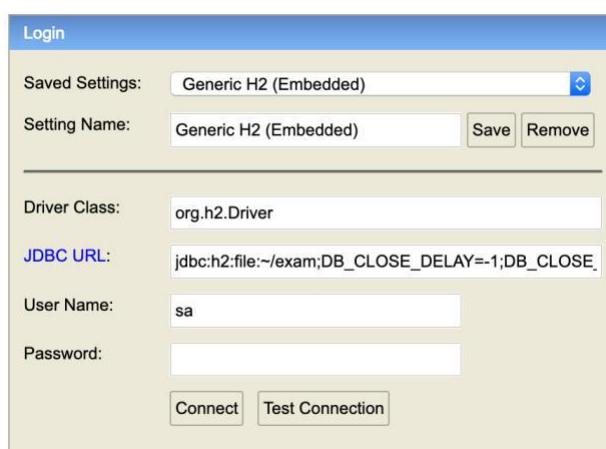
W pakiecie `pl.sdacademy.tarr.exam.ex2` znajduje się aplikacja RESTowa z przykładowo zaimplementowanymi funkcjonalnościami *Użytkownika*.

Zadania do wykonania:

### 1. Uruchom klasę `Application`

- pod adresem <http://localhost:8080/console> znajduje się konsola do zarządzania bazą danych H2, możesz tam przejrzeć schemat bazy oraz wykonywać zapytania. W pole **JDBC URL** wpisz wartość `jdbc:h2:file:~/exam;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE` - jest ona dostępna także w pliku `application.yml`. Wybierz **Connect** aby połączyć się z bazą. Dane przechowywane są na dysku w katalogu głównym w pliku `exam.mv.db`.

*Logowanie do bazy H2*



- pod adresem <http://localhost:8080/swagger-ui.html> znajduje się aplikacja *Swagger* umożliwiająca testowanie endpointów RESTowych

*Widok kontrolera `User Controller`*

user-controller : User Controller			Show/Hide	List Operations	Expand Operations
GET	/users				getUsers
POST	/users				createUser
DELETE	/users/{userId}				deleteUser
GET	/users/{userId}				getUser
PUT	/users/{userId}				updateUser

- spróbuj wykonać żądania na kontrolerze `user-controller`

### 2. Zapoznaj się z aplikacją

- w pakiecie `pl.sdacademy.tarr.exam.ex2.user` znajduje się fragment aplikacji dotyczącej *Użytkownika*
  - `User` encja reprezentująca *Użytkownika*
  - `UserRepository` repozytorium realizujące dostęp do danych
  - `UserService` klasa usługowa realizująca logikę biznesową
  - `UserController` kontroler udostępniający końcówki RESTowe

- **UserMapper** logika umożliwiająca translację pomiędzy encją a klasami DTO
- w pakiecie **pl.sdacademy.tarr.ex2.user.dto** znajdują się klasy **DTO** (Data Transfer Object)
  - **CreateUserDTO** klasa z danymi potrzebnymi do utworzenia *Użytkownika*
  - **UpdateUserDTO** klasa z danymi potrzebnymi do modyfikacji *Użytkownika*
  - **UserDTO** klasa reprezentująca *Użytkownika*

3. Na podstawie dostarczonego kodu, rozbuduj aplikację tak, by *Użytkownik* posiadał *Zadania*

- Dodaj encję **Task** posiadającą dane:
  - identyfikator, tytuł, datę utworzenia, typ oraz status
  - typy *Zadania*: BUG, TASK, FEATURE; możesz rozszerzyć o własne wartości
  - statusy *Zadania*: NEW, ACTIVE, DONE; możesz rozszerzyć o własne wartości
  - każde *Zadanie* należy do jednego *Użytkownika*
  - *Użytkownik* ma wiele *Zadań*
- Zaprojektuj relację pomiędzy *Użytkownikiem* i *Zadaniami*
  - zdecyduj, czy relacja będzie jedno, czy dwukierunkowa
- Dodaj pozostałe komponenty (repozytorium, serwis, kontroler, mapper, klasy DTO)
- Wymagania dotyczące klas DTO
  - pojedyncze *Zadanie* powinno prezentować informacje o właścicielu (*Użytkowniku*)
  - aby utworzyć *Zadanie* należy podać identyfikator właściciela (*Użytkownika*)
  - przy edycji *Zadania* nie można zmienić właściciela - posłuży do tego osobna metoda
  - przy zwracaniu pojedynczego *Użytkownika* nie ma informacji o jego *Zadaniach*
- Wymagania dotyczące kontrolera dla *Zadania*
  - tworzenie nowego *Zadania* dla podanego *Użytkownika*
  - pobieranie *Zadania* po identyfikatorze
  - pobieranie wszystkich *Zadań* danego *Użytkownika*
  - pobieranie wszystkich *Zadań* o podanym statusie danego *Użytkownika*
  - pobieranie wszystkich *Zadań* o podanym typie danego *Użytkownika*
  - pobieranie wszystkich *Zadań* o podanym statusie i typie danego *Użytkownika*
  - edycja danego *Zadania*
  - usuwanie danego *Zadania*
  - zmiana właściciela danego *Zadania*
- Zaproponuj własne dodatkowe metody (choć jedną)