Projekt systemu zoo

Praca zaliczeniowa z przedmiotu Inżynieria Oprogramowania

Cezary Regec

Czerwiec 2019

Projekt systemu zoo

Opis docelowego wdrożenia

Istotą ogrodu zoologicznego, potocznie zwanego „zoo", jest utrzymywanie terenu i infrastruktury technicznej dla żywych zwierząt, które są w nim eksponowane. Całe funkcjonowanie zoo opiera się zatem na zwierzętach, które w nim są przetrzymywane.

Zaprojektowanie systemu informatycznego metodami inżynierii oprogramowania pozwala na opracowanie rozwiązania, które będzie można wdrożyć zaspokajając wymagania postawione przez klienta. Analiza problemu rozpoczynająca się od pojęć biznesowych, a na technicznych kończąc, umożliwia usprawnienie procesu dostarczania oprogramowania. Usprawnienia te polegają m.in. na podniesieniu jakości poprzez dogłębną analizę biznesową, dostosowanie i zaprojektowanie rozwiązań adekwatnych do dostępnych środków (biorąc również pod uwagę wyniki analiz), a także wyznaczanie atomicznych zadań z wartością dla klienta, które zapewniają realne wyznaczniki postępu.

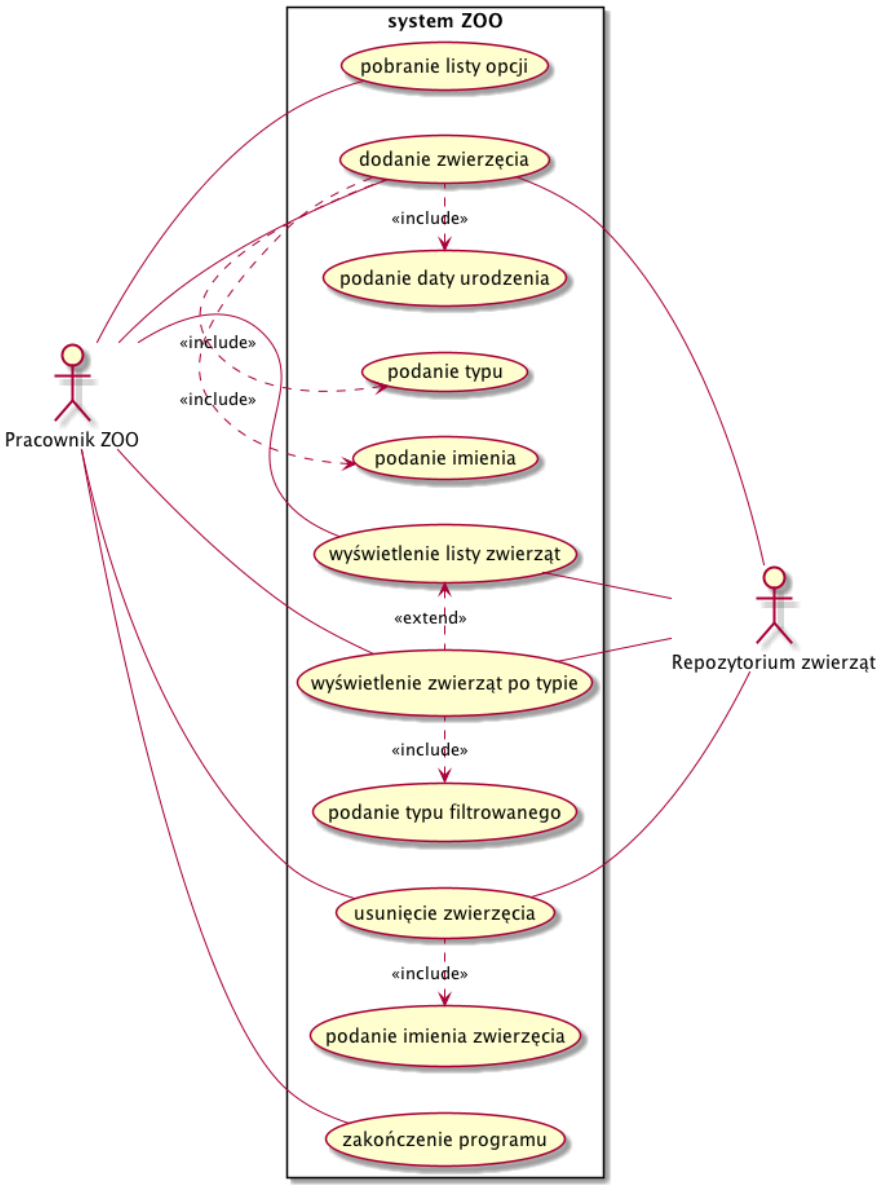
Przedstawiony projekt zawiera wnioski z analizy problemu ogrodu zoologicznego oraz wypracowane rozwiązanie z kodem źródłowym, przykładem działania oraz raportem z testów BDD (end-to-end).

Projekt ten pozwala na budowę rozszerzalnego systemu do zarządzania zoo z możliwością przechowywania informacji o zwierzętach, które są w nim przetrzymywane. System ten dostarcza podstawowej funkcjonalności dodawania, przeglądania oraz usuwania zwierząt. Dane przechowywane są w repozytorium, które pozwala na podłączenie dowolnego źródła danych.

Operacje dostępne do wykonania w aplikacji, mają swoje przypisane indeksy. Te indeksy są mapowane na odpowiednie klasy realizujące daną funkcjonalność. Takie rozwiązanie pozwala na podpięcie dowolnego interfejsu, którym domyślnie jest konsola. Ta cecha systemu wykorzystywana została do stworzenia testów BDD end-to-end, które jednocześnie dokumentują funkcjonalność aplikacji.

Rozwiązanie zostało zaopatrzone w diagram przypadków użycia, strukturę logiczną klas oraz opis operacji. Następnie prezentowane jest przykładowe działanie aplikacji oraz raporty z testów.

Przypadki użycia



Ryc.1. Przypadki użycia

Przypadek użycia	Pobranie listy opcji
Numer	UC.1
Opis	Pozwala na pobranie listy opcji, które dostępne są w aplikacji. Jedna opcja reprezentuje jedną funkcję.
Aktor inicjujący	Pracownik zoo
Pozostali aktorzy	-

Warunki wstępne	System zoo jest włączony
Warunki końcowe	System zoo zwrócił listę opcji
Rezultat	Pracownik zoo uzyskał listę opcji
Scenariusz główny	1. Pracownik zoo wybiera opcję „GET_MENU” 2. System zwraca listę dostępnych opcji

Tabela 1. Przypadek użycia UC.1

Przypadek użycia	Dodanie zwierzęcia
Numer	UC.2
Opis	Pozwala na dodanie zwierzęcia
Aktor inicjujący	Pracownik zoo
Pozostali aktorzy	Repozytorium
Warunki wstępne	System zoo jest włączony i repozytorium jest gotowe
Warunki końcowe	Repozytorium zawiera dodane zwierzę
Rezultat	Zwierzę zostało dodane do systemu
Scenariusz główny	1. Pracownik zoo wybiera opcję „ADD_ANIMAL” 2. System zaczyna procedurę pobierania parametrów 2.1.Przejście do UC.2.1 2.2.Przejście do UC.2.2 2.3.Przejście do UC.2.3 3. System stwierdza, że w repozytorium nie znajduje się zwierzę z takim imieniem. 3.1.Repozytorium przyjmuje pobrane dane zwierzęcia jako nowe zwierzę 4. System zwraca informacje o dodanym do repozytorium zwierzęciu
Scenariusz alternatywny	3. System stwierdza, że w repozytorium znajduje się już zwierzę z takim imieniem. 3.1.Poprzednie dane w repozytorium zastępowane są nowymi.

Tabela 2. Przypadek użycia UC.2

Przypadek użycia	Podanie typu
Numer	UC.2.1
Opis	Polega na podaniu typu zwierzęcia dla celów dodania go do systemu
Aktor inicjujący	Pracownik zoo
Pozostali aktorzy	-
Warunki wstępne	Rozpoczęto procedurę pobierania parametrów do UC.2
Warunki końcowe	Użytkownik wprowadził rodzaj zwierzęcia
Rezultat	System zawiera częściową informację o dodawanym zwierzęciu

Scenariusz główny	1. System prosi o podanie typu poprzez zapytanie „Rodzaj zwierzęcia: ” jednocześnie prezentując dostępne typy. 2. Pracownik zoo wprowadza dane 3. Powrót do przypadku głównego UC.2

Tabela 3. Przypadek użycia UC.2.1

Przypadek użycia	Podanie imienia
Numer	UC.2.2
Opis	Polega na podaniu imienia zwierzęcia dla celów dodania go do systemu
Aktor inicjujący	Pracownik zoo
Pozostali aktorzy	-
Warunki wstępne	Rozpoczęto procedurę pobierania parametrów do UC.2
Warunki końcowe	Użytkownik wprowadził imię zwierzęcia
Rezultat	System zawiera częściową informację o dodawanym zwierzęciu
Scenariusz główny	1. System prosi o podanie imienia poprzez zapytanie „Imię: ” 2. Pracownik zoo wprowadza dane 3. Powrót do przypadku głównego UC.2

Tabela 4. Przypadek użycia UC.2.2

Przypadek użycia	Podanie daty urodzenia
Numer	UC.2.3
Opis	Polega na podaniu daty urodzenia zwierzęcia dla celów dodania go do systemu
Aktor inicjujący	Pracownik zoo
Pozostali aktorzy	-
Warunki wstępne	Rozpoczęto procedurę pobierania parametrów do UC.2
Warunki końcowe	Użytkownik wprowadził datę urodzenia zwierzęcia
Rezultat	System zawiera częściową informację o dodawanym zwierzęciu
Scenariusz główny	1. System prosi o podanie roku poprzez zapytanie „Rok urodzenia: ” 2. Pracownik zoo wprowadza dane 3. System prosi o podanie miesiąca poprzez zapytanie „Miesiąc urodzenia: ” 4. Pracownik zoo wprowadza dane 5. System prosi o podane dnia poprzez zapytanie „Dzień urodzenia: ” 6. Pracownik zoo wprowadza dane 7. Powrót do przypadku głównego UC.2

Tabela 5. Przypadek użycia UC.2.3

Przypadek użycia	Wyświetlenie listy zwierząt
Numer	UC.3
Opis	Pozwala na wyświetlenie listy dodanych zwierząt

Aktor inicjujący	Pracownik zoo
Pozostali aktorzy	Repozytorium
Warunki wstępne	System zoo jest włączony i repozytorium jest gotowe
Warunki końcowe	System zoo zwrócił listę zwierząt
Rezultat	Pracownik zoo uzyskał listę zwierząt, które znajdują się w systemie
Scenariusz główny	1. Pracownik zoo wybiera opcję „GET_ANIMALS” 2. System pobiera dane z repozytorium 3. System zwraca listę zwierząt

Tabela 6. Przypadek użycia UC.3

Przypadek użycia	Wyświetlenie zwierząt po typie
Numer	UC.4
Opis	Pozwala na wyświetlenie listy dodanych zwierząt filtrując po zdefiniowanym rodzaju
Aktor inicjujący	Pracownik zoo
Pozostali aktorzy	Repozytorium
Warunki wstępne	System zoo jest włączony i repozytorium jest gotowe
Warunki końcowe	System zoo zwrócił listę zwierząt należących wyłącznie do danego rodzaju
Rezultat	Pracownik zoo uzyskał listę zwierząt konkretnego rodzaju, które znajdują się w systemie
Scenariusz główny	1. Pracownik zoo wybiera opcję „GET_ANIMALS_BY_TYPE” 2. System zaczyna procedurę pobierania parametrów 2.1.Przejście do przypadku UC.4.1 3. System pobiera dane z repozytorium filtrując po typie 4. System zwraca listę zwierząt

Tabela 7. Przypadek użycia UC.4

Przypadek użycia	Podanie typu filtrowanego
Numer	UC.4.1
Opis	Polega na podaniu typu zwierzęcia dla celów filtrowania listy zwierząt
Aktor inicjujący	Pracownik zoo
Pozostali aktorzy	-
Warunki wstępne	Rozpoczęto procedurę pobierania parametrów do UC.4
Warunki końcowe	Użytkownik wprowadził rodzaj zwierzęcia
Rezultat	System zawiera informację o kryterium filtrowania

Scenariusz główny	1. System prosi o podanie typu poprzez zapytanie „Rodzaj zwierzęcia: ” jednocześnie prezentując dostępne typy. 2. Pracownik zoo wprowadza dane 3. Powrót do przypadku głównego UC.4

Tabela 8. Przypadek użycia UC.4.1

Przypadek użycia	Usunięcie zwierzęcia
Numer	UC.5
Opis	Pozwala na usunięcie zwierzęcia z systemu
Aktor inicjujący	Pracownik zoo
Pozostali aktorzy	Repozytorium
Warunki wstępne	System zoo jest włączony i repozytorium jest gotowe
Warunki końcowe	System podjął próbę usunięcia zwierzęcia z repozytorium
Rezultat	W repozytorium nie znajduje się zwierzę o danym imieniu
Scenariusz główny	1. Pracownik zoo wybiera opcję „REMOVE_ANIMAL” 2. System zaczyna procedurę pobierania parametrów 2.1.Przejście do przypadku UC.5.1 3. System podejmuje próbę usunięcia zwierzęcia z repozytorium 3.1.System stwierdza, że w repozytorium jest takie zwierzę i je usuwa. 4. System zwraca informację „Usunięto zwierzę”
Scenariusz alternatywny	3. System podejmuje próbę usunięcia zwierzęcia z repozytorium 3.1.System stwierdza, że w repozytorium nie ma takiego zwierzęcia. 4. System zwraca informację „Nie usunięto żadnego zwierzęcia”

Tabela 9. Przypadek użycia UC.5

Przypadek użycia	Podanie imienia zwierzęcia
Numer	UC.5.1
Opis	Polega na podaniu imienia zwierzęcia dla celów usunięcia go z systemu
Aktor inicjujący	Pracownik zoo
Pozostali aktorzy	-
Warunki wstępne	Rozpoczęto procedurę pobierania parametrów do UC.5
Warunki końcowe	Użytkownik wprowadził imię zwierzęcia
Rezultat	System zawiera informację o usuwanym zwierzęciu
Scenariusz główny	1. System prosi o podanie imienia poprzez zapytanie „Imię: ” 2. Pracownik zoo wprowadza dane 3. Powrót do przypadku głównego UC.5

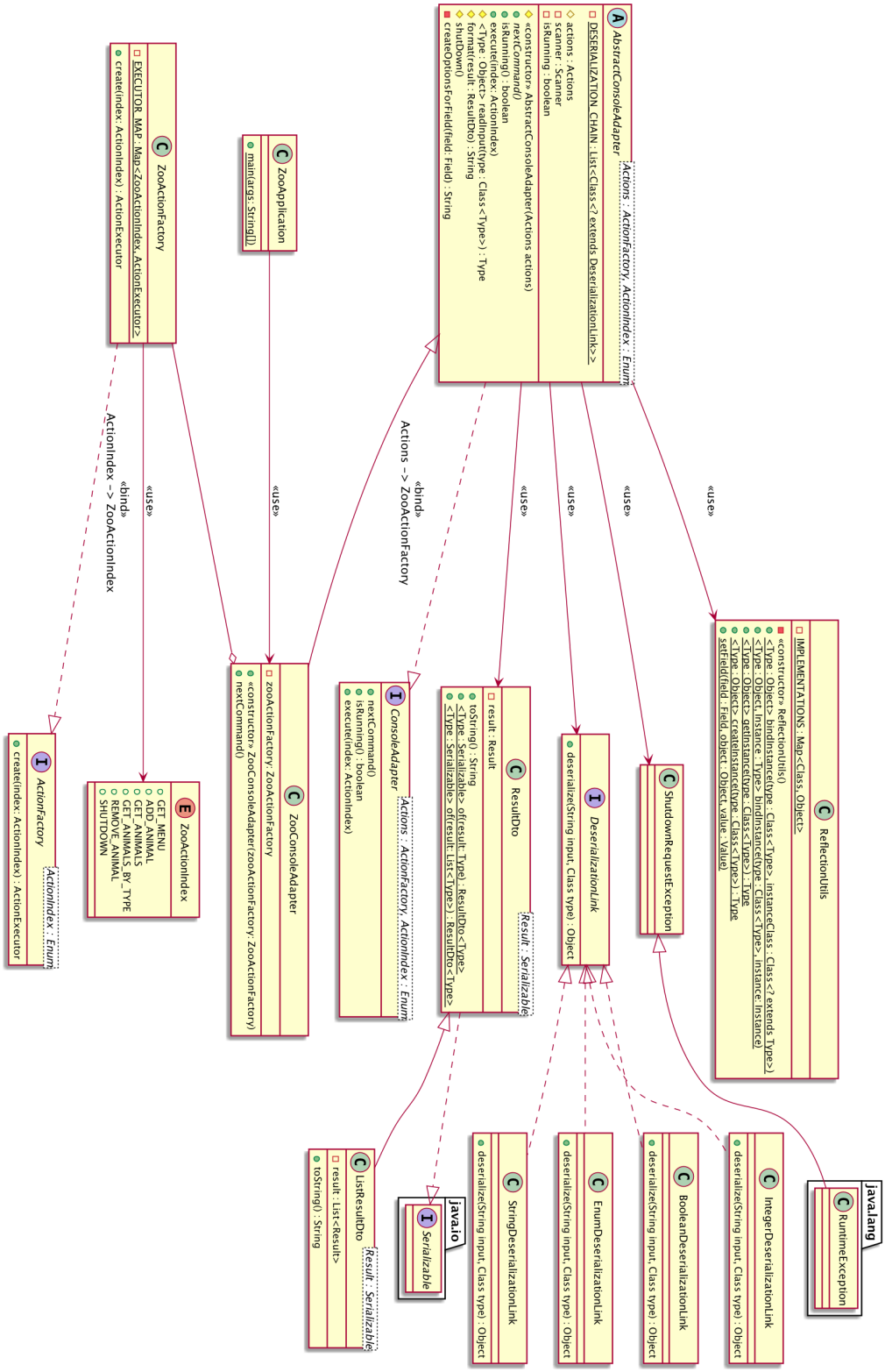
Tabela 10. Przypadek użycia UC.5.1

Przypadek użycia	Zakończenie programu
Numer	UC.6
Opis	Wyłącza system zoo
Aktor inicjujący	Pracownik zoo
Pozostali aktorzy	-
Warunki wstępne	System zoo jest włączony
Warunki końcowe	System przestaje odpowiadać na zapytania
Rezultat	System zoo jest wyłączony
Scenariusz główny	1. Pracownik zoo wybiera opcję „SHUTDOWN” 2. System kończy działanie

Tabela 11. Przypadek użycia UC.6

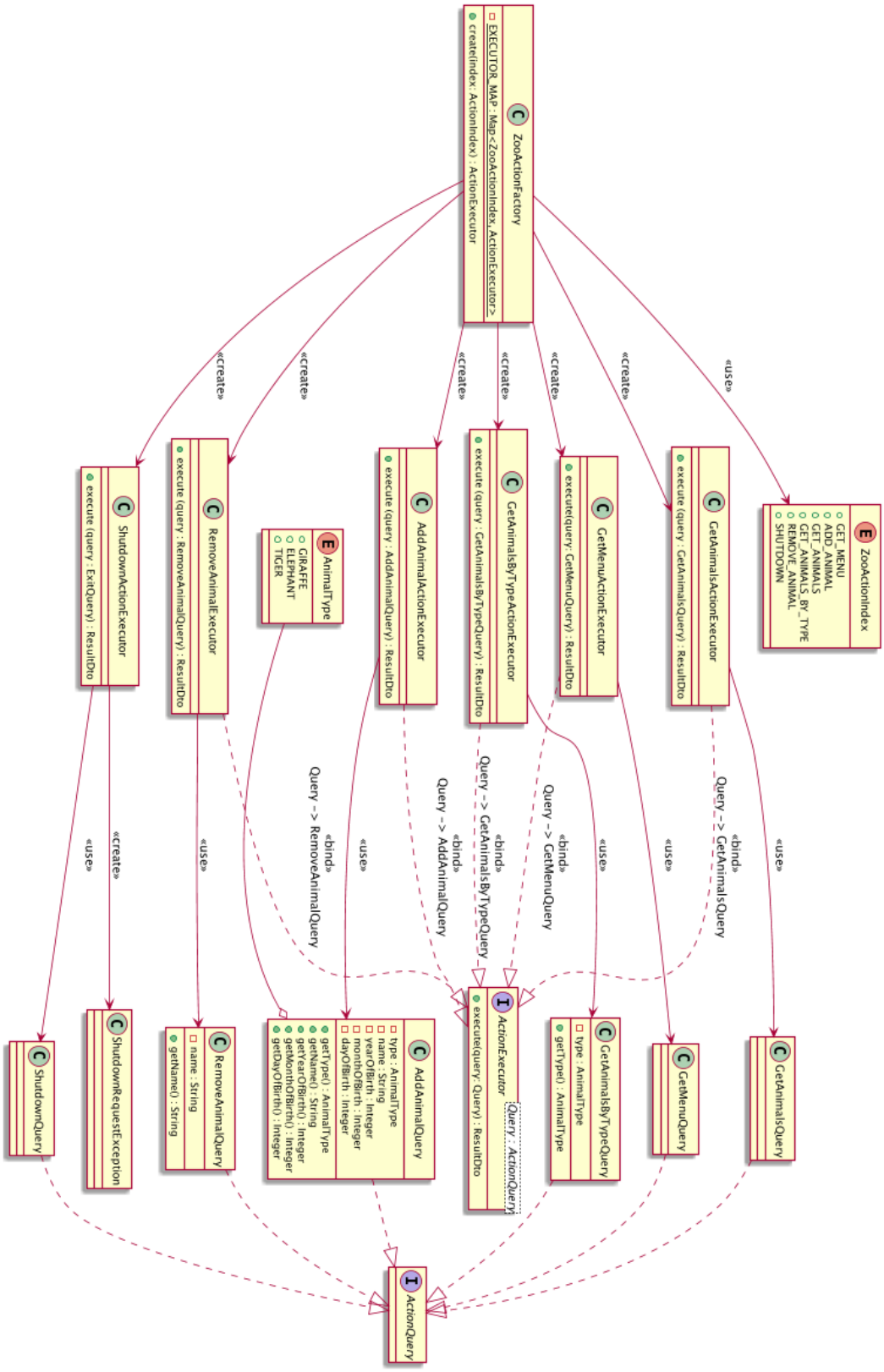
Struktura logiczna klas

Dla czytelności diagram został podzielony na części

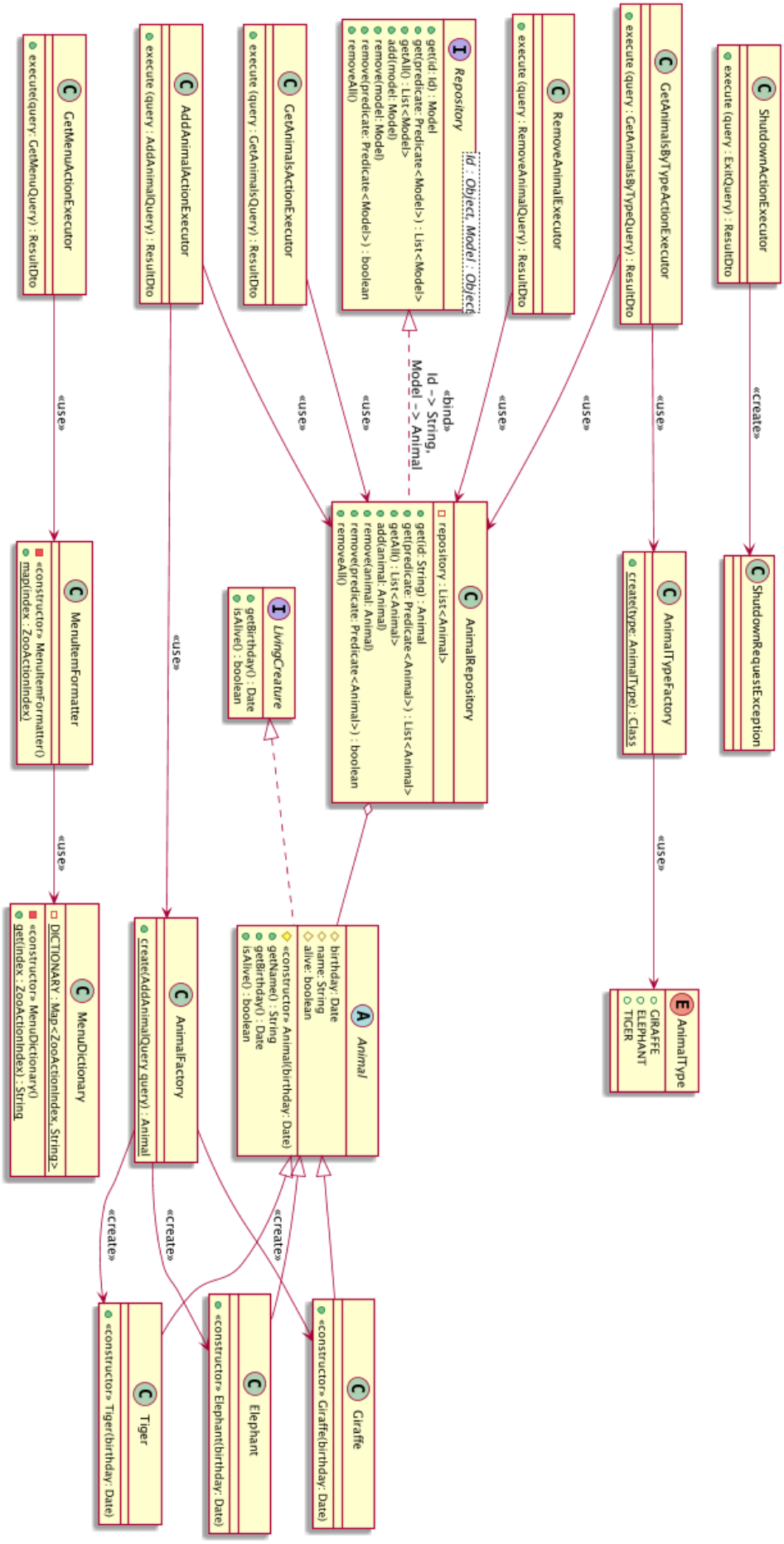


Ryc. 2. Diagram klas przedstawiający część odpowiedzialną głównie za interfejs użytkownika i przygotowanie aplikacji do użycia

Poniżej przedstawiono kolejną część systemu, która decyduje o obsłudze konkretnej opcji dostępnej w systemie, a następnie diagram klas związanych z logiką tych opcji.

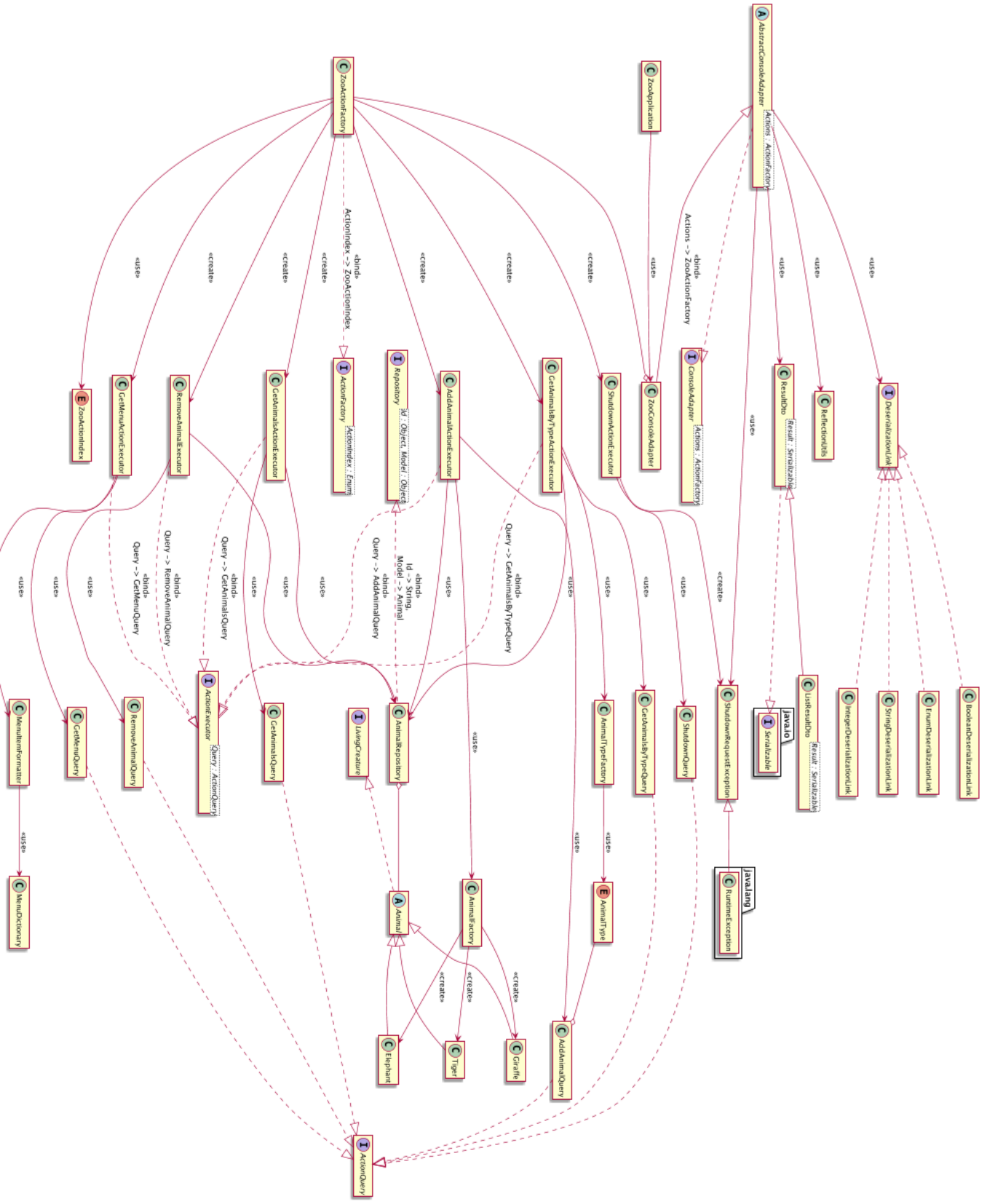


Ryc.3. Diagram klas obsługujących polecenia systemu



Ryc.4. Diagram klas wykonujących logikę poleceń

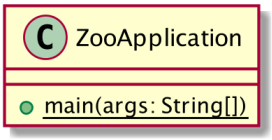
Dodatkowo poniżej znajduje się pełny diagram (bez pól i metod), który prezentuje całkowitą strukturę aplikacji.



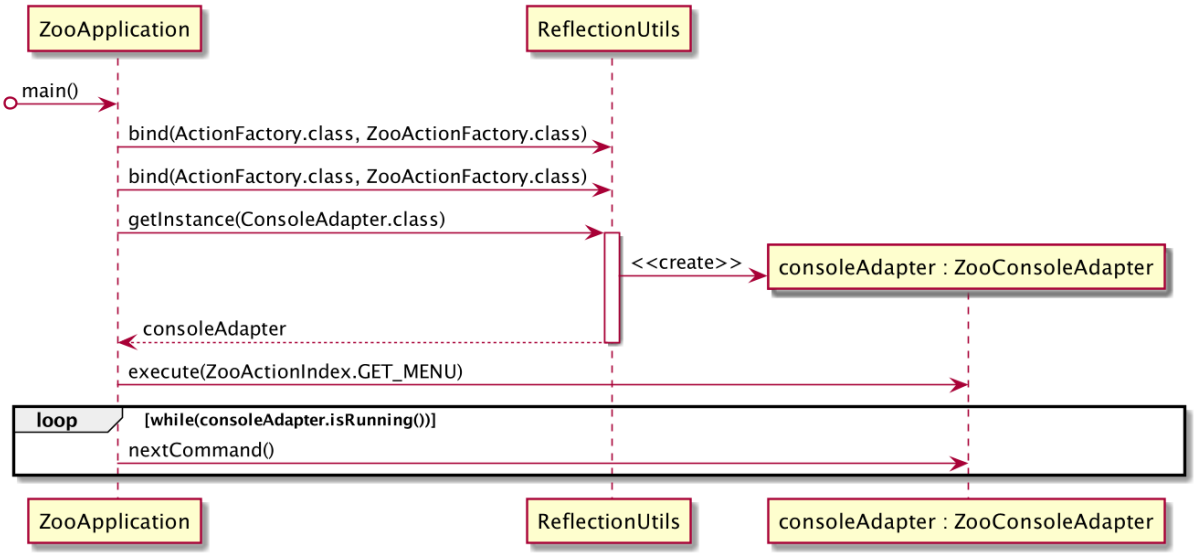
Ryc.5. Diagram wszystkich klas

Opis operacji

Klasa ZooApplication



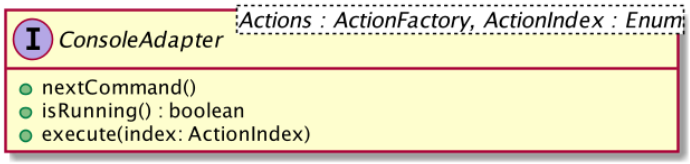
Klasa ZooApplication stanowi punkt wejścia, przygotowuje aplikację do użycia oraz przekazuje obsługę poleceń użytkownika do instancji klasy ZooConsoleAdapter.




```
package pl.cezaryregec.zoo;

import pl.cezaryregec.zoo.actions.ActionFactory;
import pl.cezaryregec.zoo.actions.ZooActionFactory;
import pl.cezaryregec.zoo.actions.ZooActionIndex;
import pl.cezaryregec.zoo.console.ConsoleAdapter;
import pl.cezaryregec.zoo.console.ZooConsoleAdapter;
import pl.cezaryregec.zoo.utils.ReflectionUtils;

public class ZooApplication {
    public static void main(String[] args) {
        ReflectionUtils.bind(ActionFactory.class, ZooActionFactory.class);
        ReflectionUtils.bind(ConsoleAdapter.class, ZooConsoleAdapter.class);

        ConsoleAdapter consoleAdapter =
ReflectionUtils.getInstance(ConsoleAdapter.class);

        consoleAdapter.execute(ZooActionIndex.GET_MENU);
```

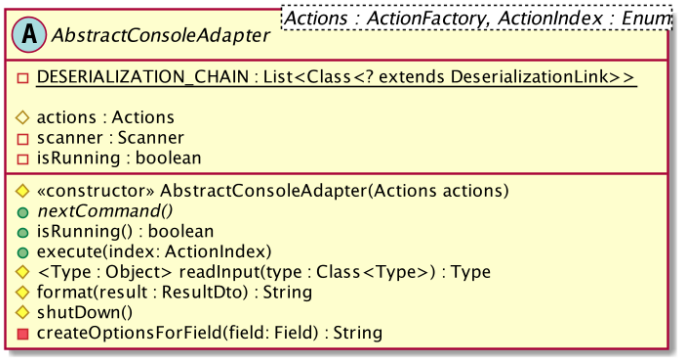
```
        while (consoleAdapter.isRunning()) {
            consoleAdapter.nextCommand();
        }
    }
}
```

Interfejs ConsoleAdapter

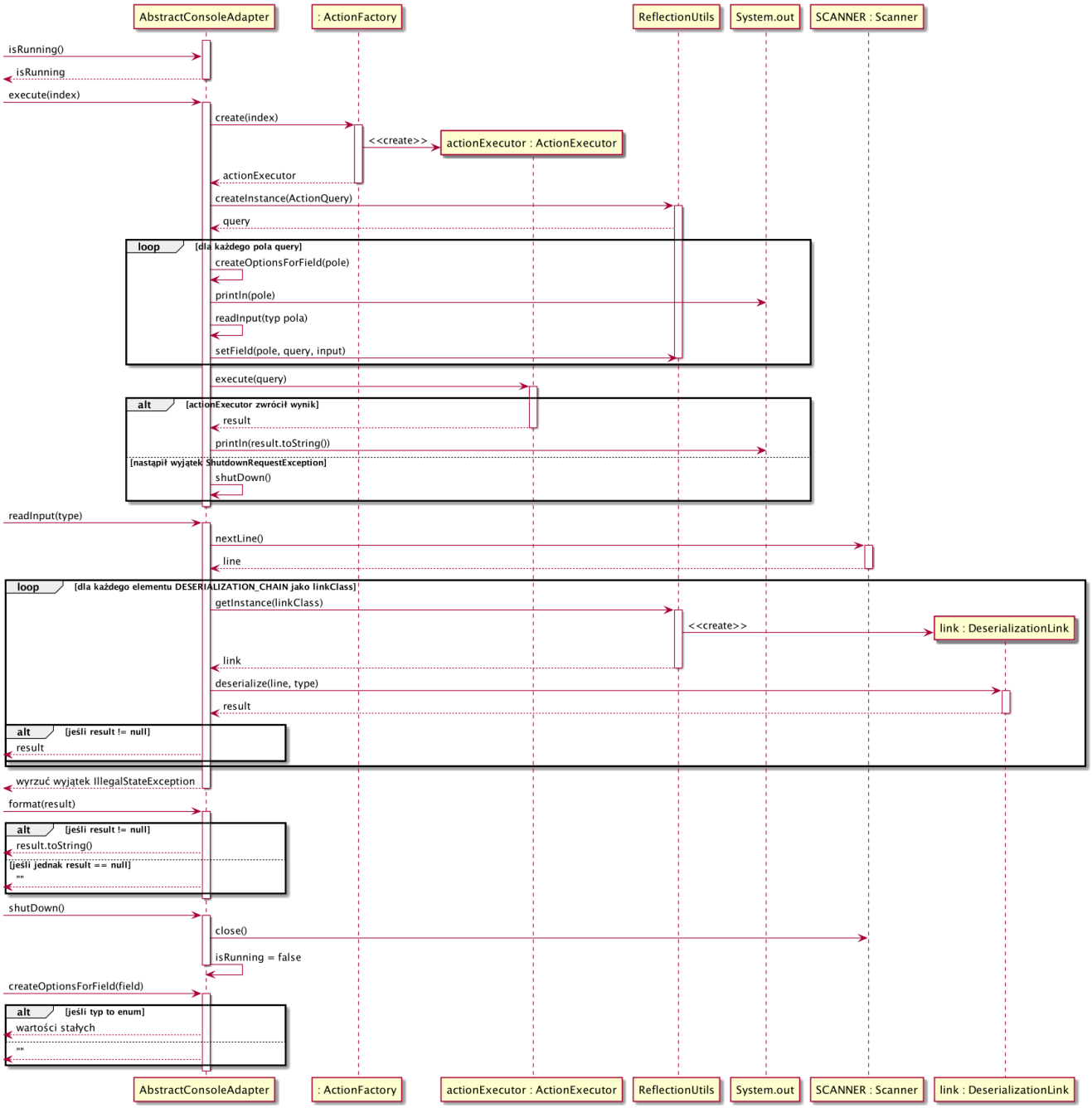


Stanowi wymaganie, które musi być spełnione przez implementację adaptera aplikacji sterowanego przez interfejs linii poleceń (konsoli).

Abstrakcyjna klasa AbstractConsoleAdapter



Jest to abstrakcyjna implementacja ConsoleAdapter zawierająca domyślne implementacje m.in. metody isRunning(), execute() oraz zawierająca metody pomocnicze, które mogą być współdzielone przez rzeczywiste implementacje ConsoleAdapter.



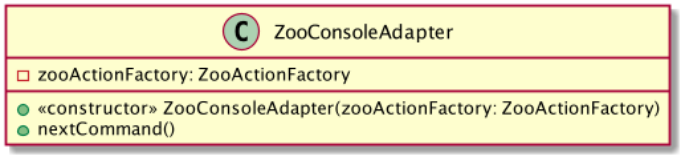

```java
package pl.cezaryregec.zoo.console;

import pl.cezaryregec.zoo.actions.*;
import pl.cezaryregec.zoo.console.annotation.ReadableName;
import pl.cezaryregec.zoo.console.deserializers.*;
import pl.cezaryregec.zoo.dto.result.ResultDto;
import pl.cezaryregec.zoo.exception.ShutdownRequestException;
import pl.cezaryregec.zoo.utils.ReflectionUtils;

import java.lang.reflect.Field;
import java.lang.reflect.Method;
import java.util.List;
import java.util.Scanner;
```

```
import java.util.stream.Collectors;
import java.util.stream.Stream;

public abstract class AbstractConsoleAdapter<Actions extends ActionFactory,
ActionIndex extends Enum<ActionIndex>> implements ConsoleAdapter<ActionIndex> {
    private static final List<Class<? extends DeserializationLink>>
DESERIALIZATION_CHAIN = Stream.of(
            StringDeserializationLink.class,
            IntegerDeserializationLink.class,
            BooleanDeserializationLink.class,
            EnumDeserializationLink.class
    ).collect(Collectors.toList());

    private final Scanner SCANNER = new Scanner(System.in);
    protected final Actions actions;
    protected boolean isRunning = true;

    protected AbstractConsoleAdapter(Actions actions) {
        this.actions = actions;
    }

    public boolean isRunning() {
        return this.isRunning;
    }

    public void shutDown() {
        this.isRunning = false;
        SCANNER.close();
    }

    public abstract void nextCommand();

    public void execute(ActionIndex index) {
        if (!isRunning) {
            throw new IllegalStateException("Console is shut down");
        }

        String executorMethodName = ActionExecutor.class.getDeclaredMethods()
[0].getName();
        Class<?> parameterType = ActionExecutor.class.getDeclaredMethods()
[0].getParameterTypes()[0];

        ActionExecutor actionExecutor = actions.create(index);
        Method[] declaredMethods = actionExecutor.getClass().getDeclaredMethods();
        Class<?> queryType = Stream.of(declaredMethods)
                .filter(method -> executorMethodName.equals(method.getName()) &&
method.getParameterTypes()[0] != parameterType)
                .map(method -> method.getParameterTypes()[0])
                .findFirst()
                .orElseThrow(() -> new IllegalStateException("Class of type " +
actionExecutor.getClass() + " is not a valid ActionExecutor"));
```

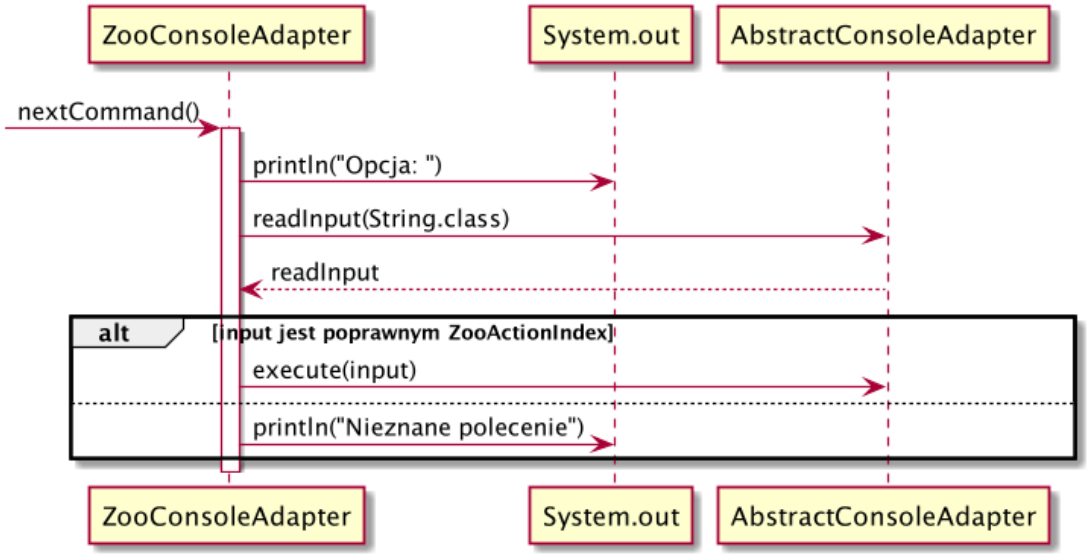
```
        ActionQuery query = (ActionQuery)
ReflectionUtils.createInstance(queryType);
        Field[] declaredFields = queryType.getDeclaredFields();

        for (Field field : declaredFields) {
            String name = field.getName();
            String options = createOptionsForField(field);
            if (field.isAnnotationPresent(ReadableName.class)) {
                name = field.getAnnotation(ReadableName.class).value();
            }

            System.out.print(name + options + ": ");
            ReflectionUtils.setField(field, query, readInput(field.getType()));
        }

        try {
            ResultDto resultDto = actionExecutor.execute(query);
            System.out.println(format(resultDto));
        } catch(ShutdownRequestException exit) {
            shutDown();
        }
    }

    private String createOptionsForField(Field field) {
        Class<?> type = field.getType();
        if (type.getSuperclass() == Enum.class) {
            List<String> constants = Stream.of(type.getDeclaredFields())
                    .map(Field::getName)
                    .filter(item -> !"$VALUES".equals(item))
                    .collect(Collectors.toList());

            return " (" + String.join(", ", constants) + ")";
        }
        return "";
    }

    protected <T> T readInput(Class<T> type) {
        String input = SCANNER.nextLine();
        for (Class<? extends DeserializationLink> linkClass :
DESERIALIZATION_CHAIN) {
            DeserializationLink link = ReflectionUtils.getInstance(linkClass);
            Object result = link.deserialize(input, type);

            if (result != null) {
                return (T) result;
            }
        }

        throw new IllegalStateException(type + " type not supported");
```

```
    }

    protected String format(ResultDto resultDto) {
        if (resultDto != null) {
            return resultDto.toString();
        }
        return "";
    }
}
```

Klasa ZooConsoleAdapter



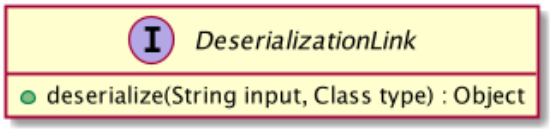
Stanowi rzeczywistą implementację ConsoleAdapter, dziedziczy wspólną funkcjonalność z AbstractConsoleAdapter i utylizuje ZooActionFactory.




```
package pl.cezaryregec.zoo.console;

import pl.cezaryregec.zoo.actions.ZooActionFactory;
import pl.cezaryregec.zoo.actions.ZooActionIndex;

public class ZooConsoleAdapter extends AbstractConsoleAdapter<ZooActionFactory,
ZooActionIndex> {
    public ZooConsoleAdapter(ZooActionFactory zooActionFactory) {
        super(zooActionFactory);
    }
```

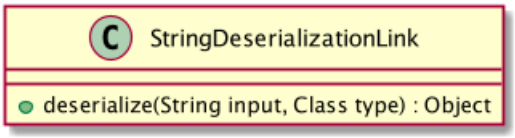
```
    @Override
    public void nextCommand() {
        if (!isRunning()) {
            throw new IllegalStateException("Console is shut down");
        }

        System.out.print("\nOpcja: ");
        String input = readInput(String.class);
        try {
            ZooActionIndex zooActionIndex = ZooActionIndex.valueOf(input);
            execute(zooActionIndex);
        } catch(IllegalArgumentException exception) {
            System.out.println("Nieznane polecenie");
        }
    }
}
```

Interfejs DeserializationLink

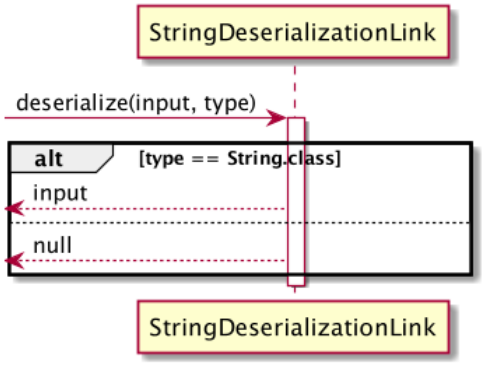


Interfejs ten stanowi wymaganie, które musi spełniać implementacja ogniwa deserializatora (klasy, która wykona deserializację w zależności od tego, czy jest w stanie lub zwróci null).

Klasa StringDeserializationLink

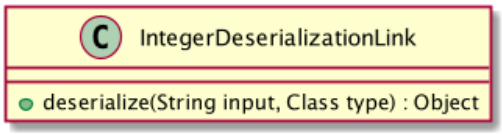


Podstawowy deserializator, dla typu String zwraca input.

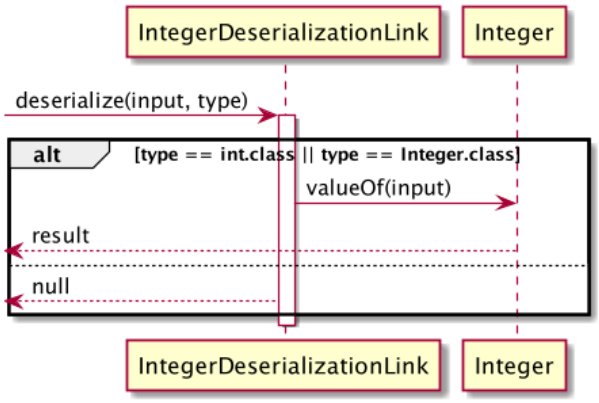


```
package pl.cezaryregec.zoo.console.deserializers;

public class StringDeserializationLink implements DeserializationLink {

    @Override
    public Object deserialize(String input, Class<?> type) {
        if (type == String.class) {
            return input;
        }

        return null;
    }
}
```

Klasa IntegerDeserializationLink

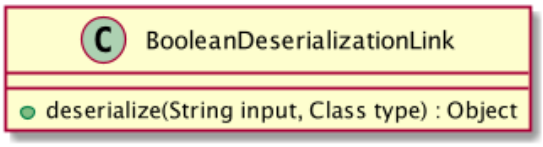


Ta klasa deserializuje w przypadku napotkania typu int bądź Integer.

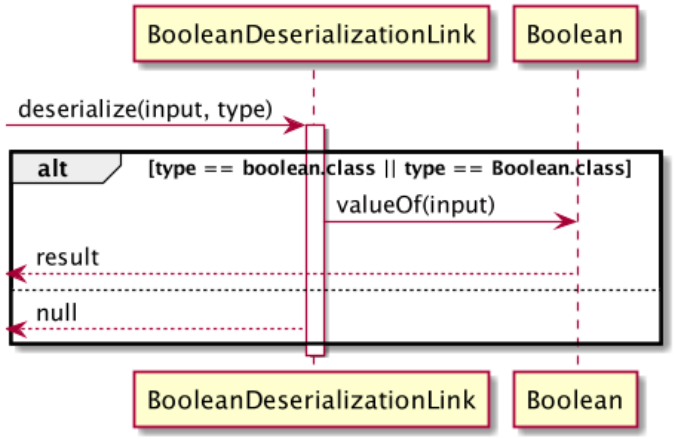



```
package pl.cezaryregec.zoo.console.deserializers;

public class IntegerDeserializationLink implements DeserializationLink {
    @Override
    public Object deserialize(String input, Class<?> type) {
        if (type == int.class || type == Integer.class) {
            return Integer.valueOf(input);
        }
        return null;
    }
}
```

Klasa BooleanDeserializationLink

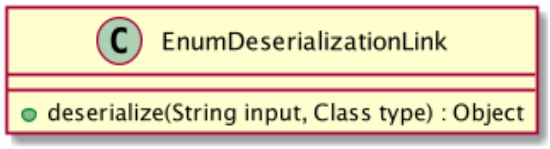


Zadaniem tej jest deserializacja w przypadku napotkania typu Boolean lub boolean.

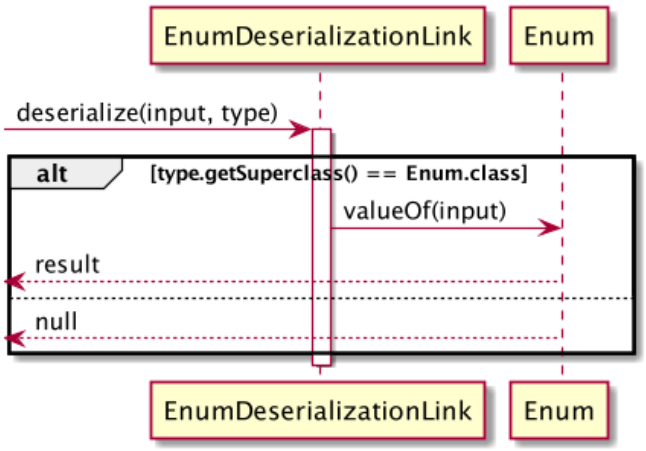



```
package pl.cezaryregec.zoo.console.deserializers;

public class BooleanDeserializationLink implements DeserializationLink {

    @Override
    public Object deserialize(String input, Class<?> type) {
        if (type == Boolean.class || type == boolean.class) {
            return Boolean.valueOf(input);
        }

        return null;
    }
}
```

Klasa EnumDeserializationLink



Zadaniem tej klasy jest deserializacja w przypadku napotkania typu pochodnego Enum.



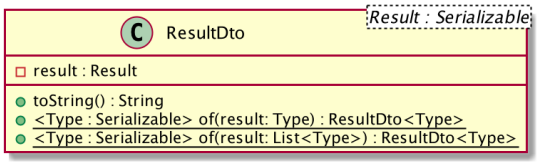

```
package pl.cezaryregec.zoo.console.deserializers;

public class EnumDeserializationLink implements DeserializationLink {
    @Override
    public Object deserialize(String input, Class<?> type) {
        if (type.getSuperclass() == Enum.class) {
            Class<? extends Enum> enumClass = (Class<? extends Enum>) type;
            return Enum.valueOf(enumClass, input);
        }
        return null;
    }
}
```

Adnotacja ReadableName

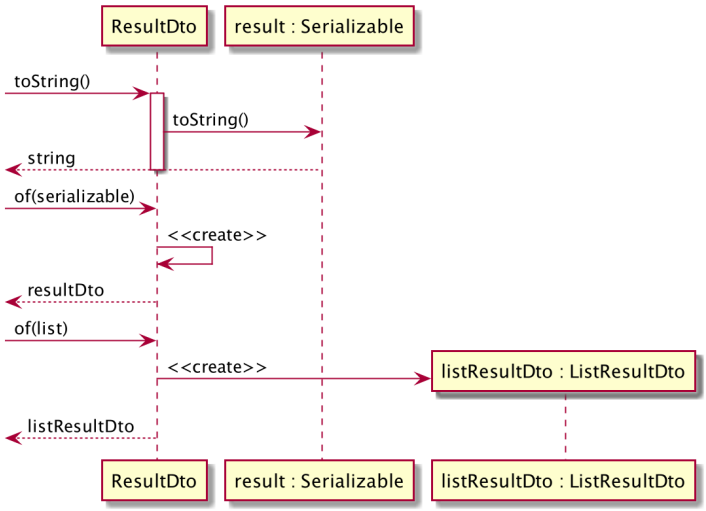
Pomocnicza adnotacja do nadawania polom czytelnych nazw, które będą prezentowane użytkownikowi.

```
package pl.cezaryregec.zoo.console.annotation;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.FIELD)
public @interface ReadableName {
    String value();
}
```

Klasa ResultDto



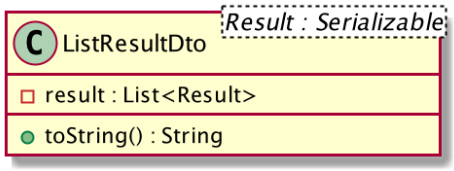
Klasa ta przechowuje wynik operacji i zawiera dwie metody statyczne tworzące obiekt ResultDto (dla pojedynczego rezultatu oraz dla listy).



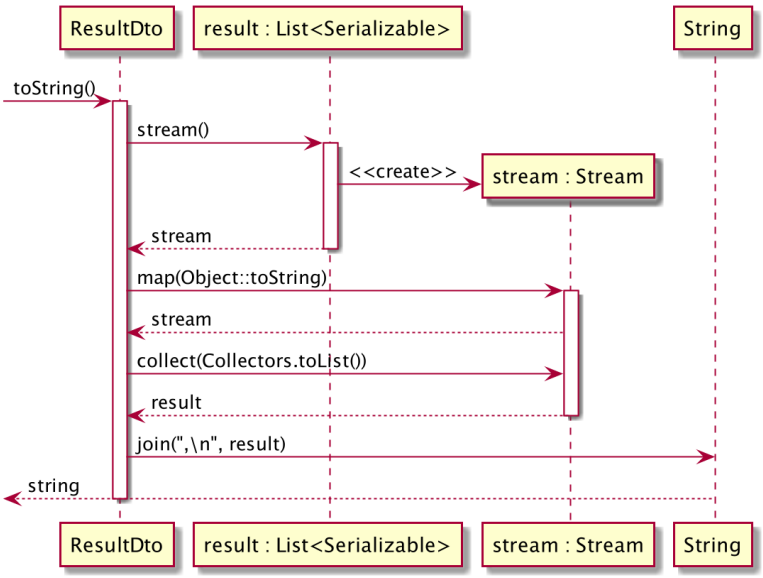

```java
package pl.cezaryregec.zoo.dto.result;

import java.io.Serializable;
import java.util.List;

public class ResultDto<Result extends Serializable> {
    private final Result result;

    public ResultDto(Result result) {
        this.result = result;
    }

    @Override
    public String toString() {
        return result.toString();
    }

    public static <T extends Serializable> ResultDto<T> of(List<T> list) {
        return new ListResultDto<>(list);
    }

    public static <T extends Serializable> ResultDto<T> of(T result) {
```

```
        return new ResultDto<T>(result);
    }
}
```

Klasa ListResultDto



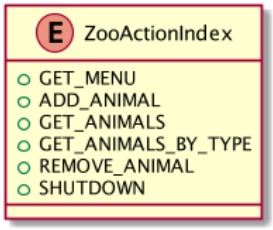
Ta klasa rozszerza ResultDto i przechowuje wynik operacji w postaci listy. Rozdzielenie tych klas pozwala na nadpisanie logiki toString() taką, która obsługiwałaby listę.




```
package pl.cezaryregec.zoo.dto.result;

import java.io.Serializable;
import java.util.List;
import java.util.stream.Collectors;

public class ListResultDto<Result extends Serializable> extends ResultDto<Result> {
    private final List<Result> result;

    public ListResultDto(List<Result> result) {
        super(null);
        this.result = result;
    }
```

```
    @Override
    public String toString() {
        List<String> stringResultList = result.stream()
                .map(Object::toString)
                .collect(Collectors.toList());
        return String.join("\n", stringResultList);
    }
}
```

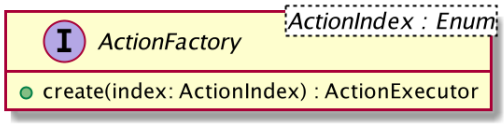
Typ wyliczeniowy ZooActionIndex



Przechowuje stałe oznaczające operacje dostępne w systemie.

```
public enum ZooActionIndex {
    GET_MENU,
    ADD_ANIMAL,
    GET_ANIMALS,
    GET_ANIMALS_BY_TYPE,
    REMOVE_ANIMAL,
    SHUTDOWN;
}
```

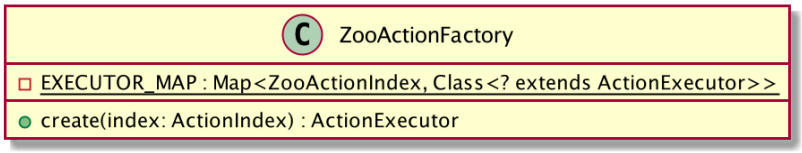
Interfejs ActionFactory



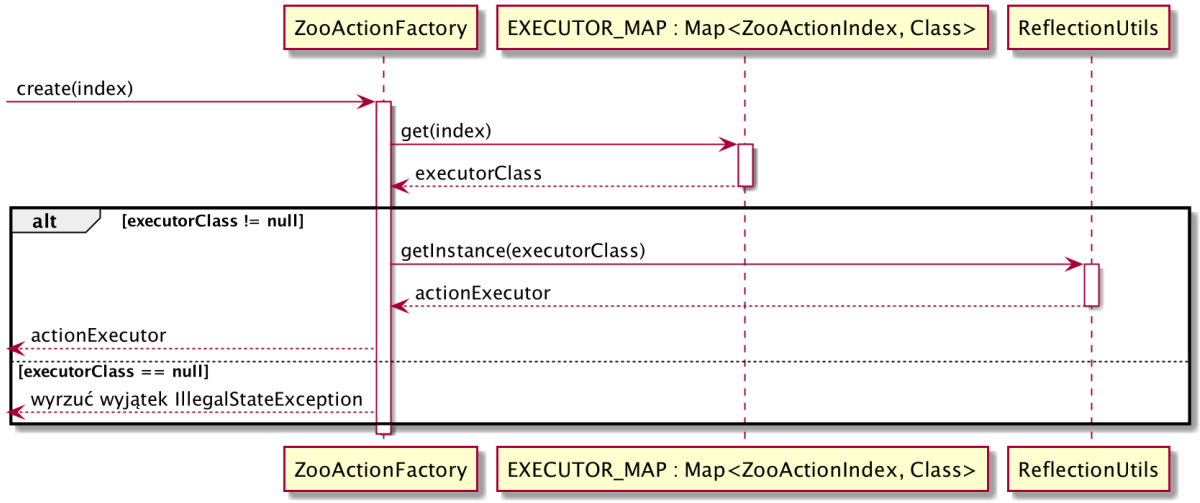
Stanowi wymaganie, które musi zostać spełnione przez fabrykę implementacji ActionExecutor.

```
package pl.cezaryregec.zoo.actions;

public interface ActionFactory<ActionIndex extends Enum<ActionIndex>> {
    ActionExecutor create(ActionIndex index);
}
```

Klasa ZooActionFactory



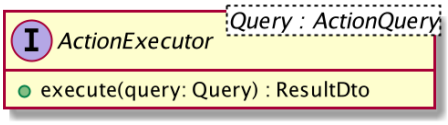
Stanowi implementację ActionFactory dla systemu zoo – skojarzona jest z operacjami określonymi przez ZooActionIndex oraz zawiera mapę tych indeksów skojarzoną z klasami (implementacji ActionExecutor). Jest fabryką implementacji ActionExecutor dla konkretnych operacji.




```
package pl.cezaryregec.zoo.actions;

import pl.cezaryregec.zoo.actions.animals.add.AddAnimalActionExecutor;
import pl.cezaryregec.zoo.actions.animals.get.GetAnimalsActionExecutor;
import pl.cezaryregec.zoo.actions.animals.get.GetAnimalsByTypeActionExecutor;
import pl.cezaryregec.zoo.actions.animals.remove.RemoveAnimalActionExecutor;
import pl.cezaryregec.zoo.actions.shutdown.ShutdownActionExecutor;
import pl.cezaryregec.zoo.actions.menu.GetMenuActionExecutor;
import pl.cezaryregec.zoo.utils.ReflectionUtils;

import java.util.HashMap;
import java.util.Map;

public class ZooActionFactory implements ActionFactory<ZooActionIndex> {
    private static final Map<ZooActionIndex, Class<? extends ActionExecutor>>
EXECUTOR_MAP = new HashMap<>();

    static {
        EXECUTOR_MAP.put(ZooActionIndex.GET_MENU, GetMenuActionExecutor.class);
        EXECUTOR_MAP.put(ZooActionIndex.ADD_ANIMAL, AddAnimalActionExecutor.class);
        EXECUTOR_MAP.put(ZooActionIndex.GET_ANIMALS,
GetAnimalsActionExecutor.class);
```

```
        EXECUTOR_MAP.put(ZooActionIndex.GET_ANIMALS_BY_TYPE,
GetAnimalsByTypeActionExecutor.class);
        EXECUTOR_MAP.put(ZooActionIndex.REMOVE_ANIMAL,
RemoveAnimalActionExecutor.class);
        EXECUTOR_MAP.put(ZooActionIndex.SHUTDOWN, ShutdownActionExecutor.class);
    }

    public ActionExecutor create(ZooActionIndex zooActionIndex) {
        Class<? extends ActionExecutor> executorClass =
EXECUTOR_MAP.get(zooActionIndex);

        if (executorClass != null) {
            return ReflectionUtils.getInstance(executorClass);
        }

        throw new IllegalStateException(String.format("%s is not supported",
zooActionIndex));
    }
}
```

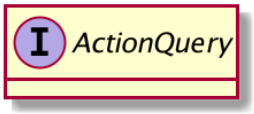
Interfejs ActionExecutor



Interfejs stanowi wymaganie, które musi spełniać klasa, która realizuje logikę konkretnej operacji obsługiwanej przez system.

```
package pl.cezaryregec.zoo.actions;

import pl.cezaryregec.zoo.dto.result.ResultDto;

public interface ActionExecutor<Query extends ActionQuery> {
    ResultDto execute(Query query);
}
```

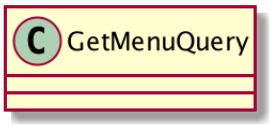
Interfejs ActionQuery



Interfejs stanowi wymaganie dla klasy agregującej parametry danej operacji.

```
package pl.cezaryregec.zoo.actions;

public interface ActionQuery {}
```

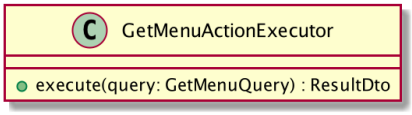
Klasa GetMenuQuery



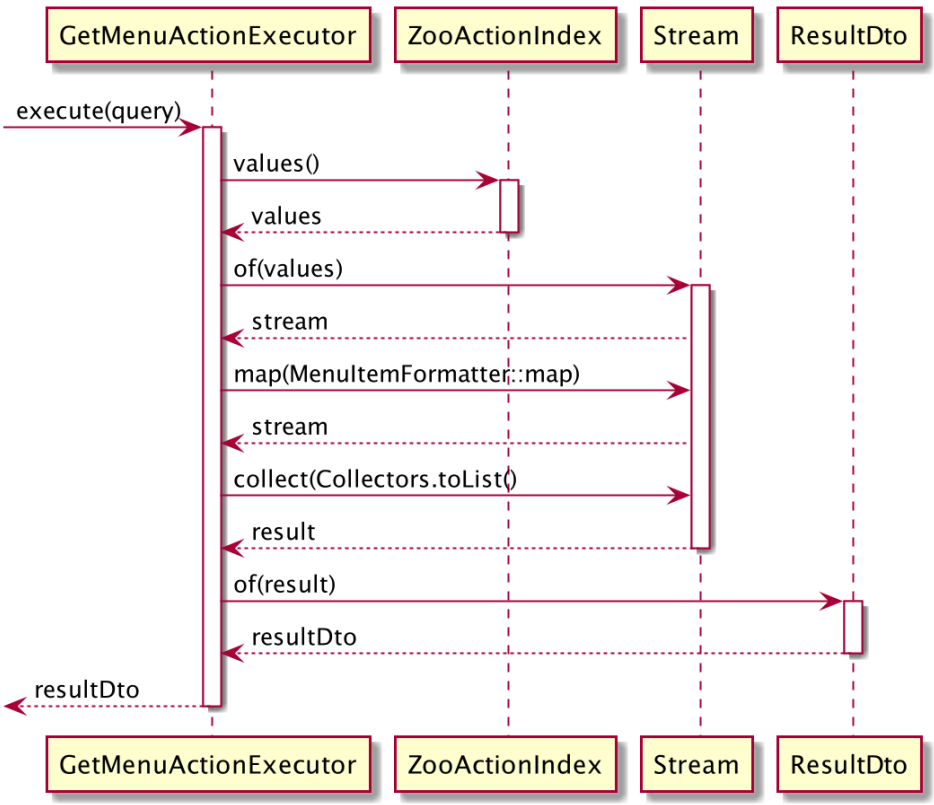
Klasa GetMenuQuery jest klasą agregującą parametry operacji GET_MENU — ta operacja nie wymaga żadnych parametrów. Klasa wykorzystuje adnotację biblioteki Lombok @NoArgsConstructor dla uproszczenia formy.

```
package pl.cezaryregec.zoo.actions.menu;

import lombok.NoArgsConstructor;
import pl.cezaryregec.zoo.actions.ActionQuery;

@NoArgsConstructor
public class GetMenuQuery implements ActionQuery {}
```

Klasa GetMenuActionExecutor

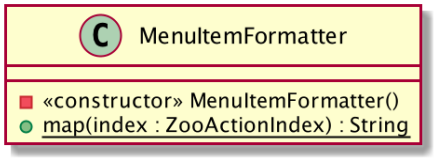


Klasa GetMenuActionExecutor wykonuje logikę operacji GET_MENU.

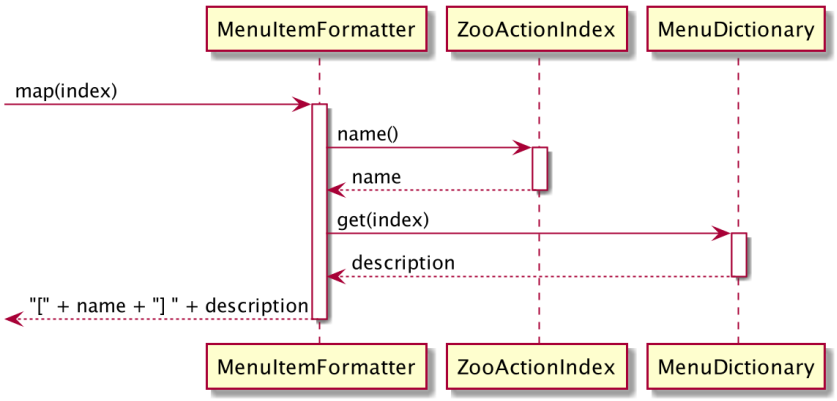


```java
package pl.cezaryregec.zoo.actions.menu;

import pl.cezaryregec.zoo.actions.ActionExecutor;
import pl.cezaryregec.zoo.actions.ZooActionIndex;
import pl.cezaryregec.zoo.dto.result.ListResultDto;
import pl.cezaryregec.zoo.dto.result.ResultDto;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class GetMenuActionExecutor implements ActionExecutor<GetMenuQuery> {
    @Override
    public ResultDto execute(GetMenuQuery query) {
        List<String> result = Stream.of(ZooActionIndex.values())
                .map(MenuItemFormatter::map)
                .collect(Collectors.toList());

        return ResultDto.of(result);
    }
}
```

Klasa MenuItemFormatter

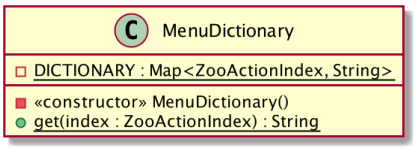


To pomocnicza klasa zawierająca jedynie statyczną metodę map, która mapuje ZooActionIndex na sformatowany element menu.

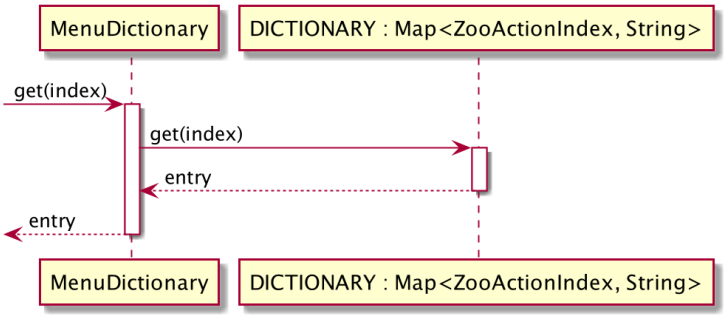


```
package pl.cezaryregec.zoo.actions.menu;

import pl.cezaryregec.zoo.actions.ZooActionIndex;

class MenuItemFormatter {
    private MenuItemFormatter() {}

    static String map(ZooActionIndex index) {
        return "[" + index.name() + "] " + MenuDictionary.get(index);
    }
}
```

Klasa MenuDictionary



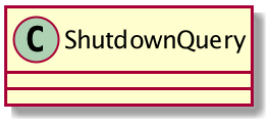
Ta pomocnicza klasa przechowuje opisy wszystkich operacji dostępnych w systemie.




```
package pl.cezaryregec.zoo.actions.menu;

import pl.cezaryregec.zoo.actions.ZooActionIndex;

import java.util.HashMap;
import java.util.Map;

class MenuDictionary {
    private static final Map<ZooActionIndex, String> DICTIONARY = new HashMap<>();
    static {
        DICTIONARY.put(ZooActionIndex.GET_MENU, "Menu dostępnych opcji");
        DICTIONARY.put(ZooActionIndex.ADD_ANIMAL, "Dodaj zwierzę");
        DICTIONARY.put(ZooActionIndex.GET_ANIMALS, "Wyświetl zwierzęta w ZOO");
        DICTIONARY.put(ZooActionIndex.GET_ANIMALS_BY_TYPE, "Wyświetl zwierzęta w
ZOO filtrując po typie");
        DICTIONARY.put(ZooActionIndex.REMOVE_ANIMAL, "Usuń zwierzę");
        DICTIONARY.put(ZooActionIndex.SHUTDOWN, "Zakończ program");
    }
```

```
    private MenuDictionary() {}

    static String get(ZooActionIndex index) {
        return DICTIONARY.get(index);
    }
}
```

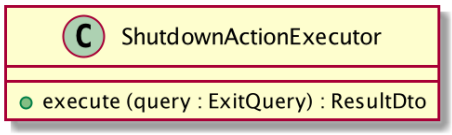
Klasa ShutdownQuery



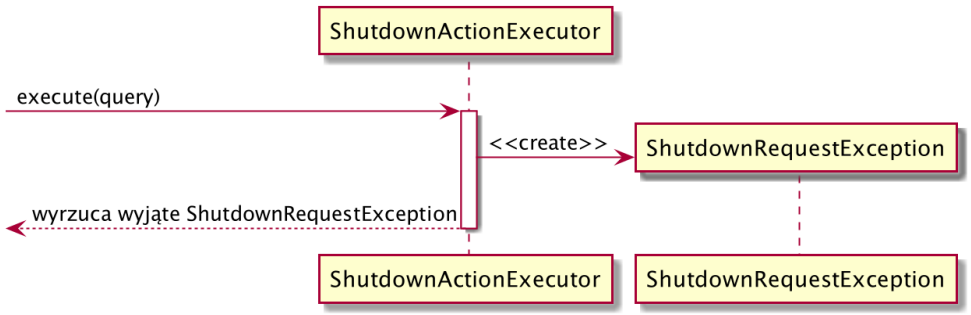
Klasa ShutdownQuery jest klasą agregującą parametry operacji SHUTDOWN — ta operacja nie wymaga żadnych parametrów. Klasa wykorzystuje adnotację biblioteki Lombok @NoArgsConstructor dla uproszczenia formy.

```
package pl.cezaryregec.zoo.actions.shutdown;

import lombok.NoArgsConstructor;
import pl.cezaryregec.zoo.actions.ActionQuery;

@NoArgsConstructor
public class ShutdownQuery implements ActionQuery {}
```

Klasa ShutdownActionExecutor



Klasa ta ma za zadanie obsłużyć operację SHUTDOWN poprzez wyrzucenie wyjątku żądania zakończenia aplikacji.



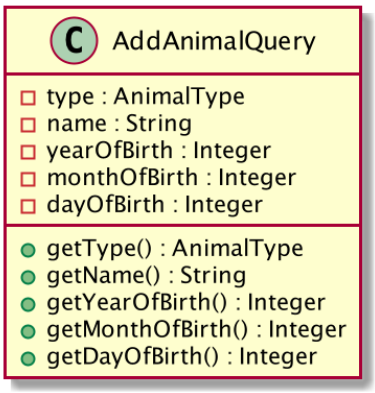
```java
package pl.cezaryregec.zoo.actions.shutdown;

import pl.cezaryregec.zoo.actions.ActionExecutor;
import pl.cezaryregec.zoo.dto.result.ResultDto;
import pl.cezaryregec.zoo.exception.ShutdownRequestException;

public class ShutdownActionExecutor implements ActionExecutor<ShutdownQuery> {
    @Override
    public ResultDto execute(ShutdownQuery query) {
        throw new ShutdownRequestException();
    }
}
```

Klasa ShutdownRequestException

Klasa ta rozszerza klasę RuntimeException i ma za zadanie sygnalizować żądanie zakończenia działania systemu.

```java
package pl.cezaryregec.zoo.exception;

public class ShutdownRequestException extends RuntimeException {}
```

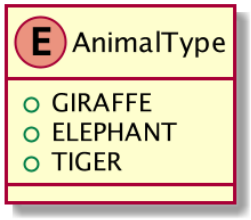
Klasa AddAnimalQuery



Klasa AddAnimalQuery jest klasą agregującą parametry operacji ADD_ANIMAL. Klasa wykorzystuje adnotacje biblioteki Lombok dla uproszczenia formy. Dodatkowo zastosowanie tutaj ma @ReadableName.

```java
package pl.cezaryregec.zoo.actions.animals.add;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Getter;
import lombok.NoArgsConstructor;
import pl.cezaryregec.zoo.actions.ActionQuery;
import pl.cezaryregec.zoo.actions.animals.AnimalType;
import pl.cezaryregec.zoo.console.annotation.ReadableName;

@Builder
@NoArgsConstructor
@AllArgsConstructor
@Getter
public class AddAnimalQuery implements ActionQuery {
    @ReadableName("Rodzaj zwierzęcia")
    private AnimalType type;

    @ReadableName("Imię zwierzęcia")
    private String name;

    @ReadableName("Rok urodzenia")
    private Integer yearOfBirth;

    @ReadableName("Miesiąc urodzenia")
    private Integer monthOfBirth;

    @ReadableName("Dzień urodzenia")
    private Integer dayOfBirth;
}
```

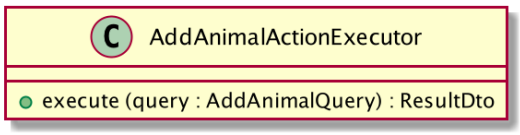
Typ wyliczeniowy AnimalType



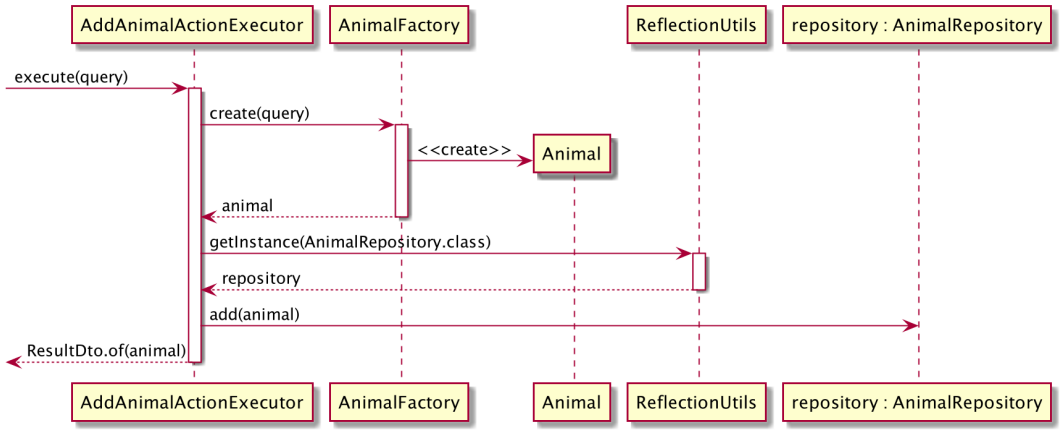
Ten typ wyliczeniowy przechowuje dostępne rodzaje zwierząt, które obsługuje system.

```java
package pl.cezaryregec.zoo.actions.animals;

public enum AnimalType {
    GIRAFFE,
    ELEPHANT,
    TIGER
}
```

Klasa AddAnimalActionExecutor

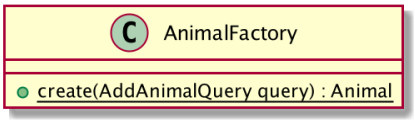


Ta klasa realizuje logikę operacji dodania zwierzęcia do repozytorium (operacja ADD_ANIMAL).

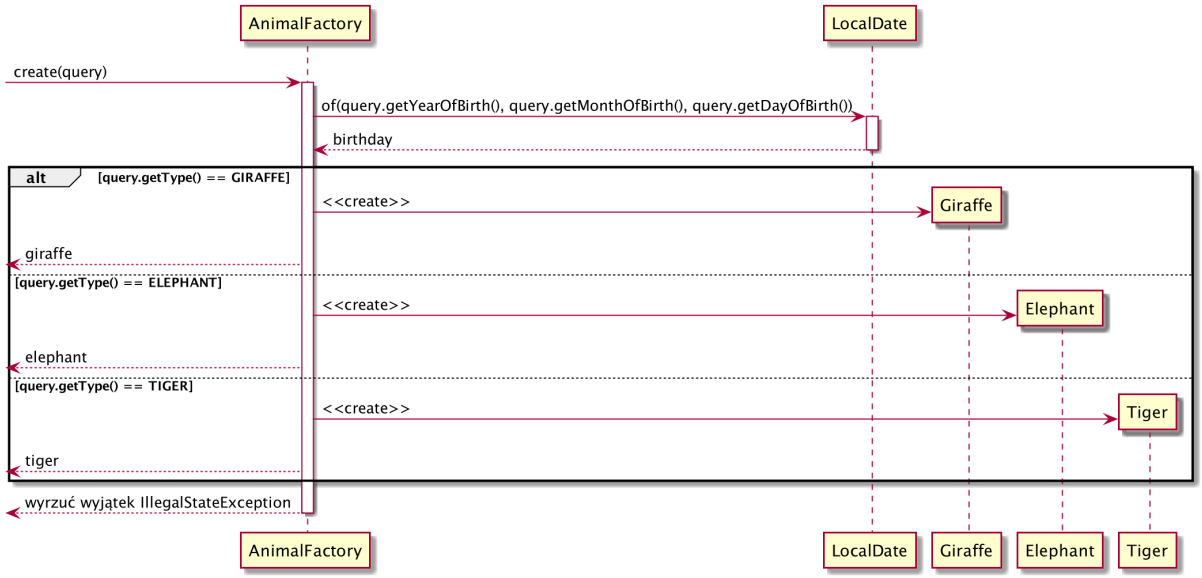



```java
package pl.cezaryregec.zoo.actions.animals.add;

import pl.cezaryregec.zoo.actions.ActionExecutor;
import pl.cezaryregec.zoo.dto.result.ResultDto;
import pl.cezaryregec.zoo.model.animal.Animal;
import pl.cezaryregec.zoo.repository.AnimalRepository;
import pl.cezaryregec.zoo.utils.ReflectionUtils;

public class AddAnimalActionExecutor implements ActionExecutor<AddAnimalQuery> {
    @Override
    public ResultDto execute(AddAnimalQuery query) {
        Animal animal = AnimalFactory.create(query);
        AnimalRepository animalRepository =
ReflectionUtils.getInstance(AnimalRepository.class);
        animalRepository.add(animal);
        return ResultDto.of(animal);
    }
}
```

Klasa AnimalFactory



Ta klasa jest statyczną fabryką obiektów Animal, m.in. Giraffe, Elephant i Tiger.




```
package pl.cezaryregec.zoo.actions.animals.add;

import pl.cezaryregec.zoo.model.animal.Animal;
import pl.cezaryregec.zoo.model.animal.Elephant;
import pl.cezaryregec.zoo.model.animal.Giraffe;
import pl.cezaryregec.zoo.model.animal.Tiger;

import java.time.LocalDate;

class AnimalFactory {
    static Animal create(AddAnimalQuery query) {
        LocalDate birthday = LocalDate.of(query.getYearOfBirth(),
query.getMonthOfBirth(), query.getDayOfBirth());
        switch (query.getType()) {
            case GIRAFFE:
                return new Giraffe(query.getName(), birthday);
            case ELEPHANT:
                return new Elephant(query.getName(), birthday);
            case TIGER:
                return new Tiger(query.getName(), birthday);
        }
        throw new IllegalStateException("Unknown animal type");
    }
}
```

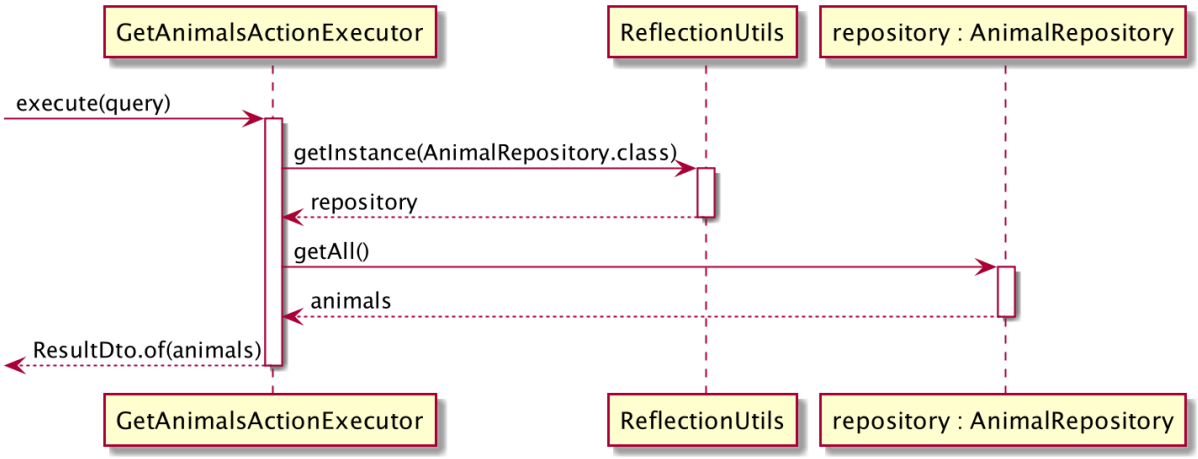
Klasa GetAnimalsQuery

Klasa GetAnimalsQuery jest klasą agregującą parametry operacji GET_ANIMALS — ta operacja nie wymaga żadnych parametrów.

```
package pl.cezaryregec.zoo.actions.animals.get;

import lombok.NoArgsConstructor;
import pl.cezaryregec.zoo.actions.ActionQuery;

@NoArgsConstructor
public class GetAnimalsQuery implements ActionQuery {}
```

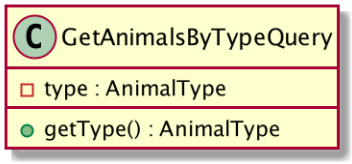
Klasa GetAnimalsActionExecutor

Ta klasa realizuje logikę operacji wyświetlenia zwierząt znajdujących się w repozytorium (operacja GET_ANIMALS).




```
package pl.cezaryregec.zoo.actions.animals.get;

import pl.cezaryregec.zoo.actions.ActionExecutor;
import pl.cezaryregec.zoo.dto.result.ResultDto;
import pl.cezaryregec.zoo.repository.AnimalRepository;
import pl.cezaryregec.zoo.utils.ReflectionUtils;

public class GetAnimalsActionExecutor implements ActionExecutor<GetAnimalsQuery> {
    @Override
    public ResultDto execute(GetAnimalsQuery query) {
        AnimalRepository repository =
ReflectionUtils.getInstance(AnimalRepository.class);
        return ResultDto.of(repository.getAll());
    }
}
```

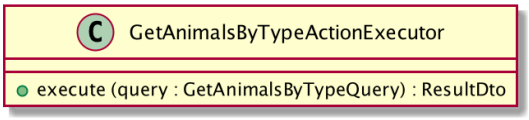
Klasa GetAnimalsByTypeQuery



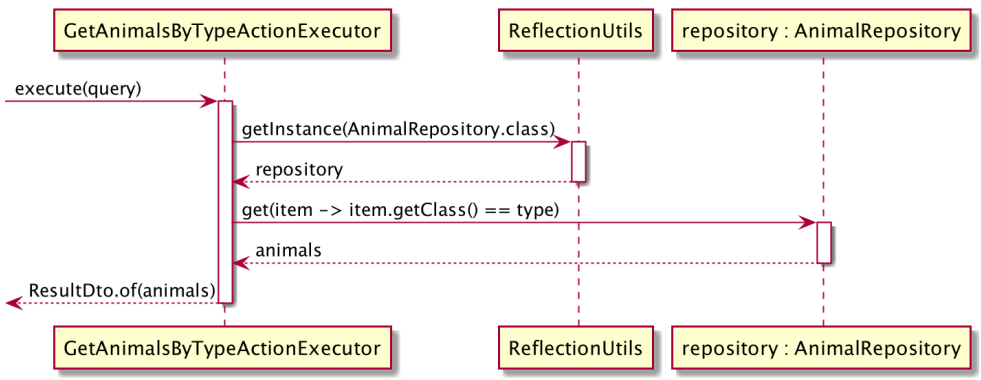
Klasa GetAnimalsByTypeQuery jest klasą agregującą parametry operacji GET_ANIMALS_BY_TYPE. Klasa wykorzystuje adnotacje biblioteki Lombok dla uproszczenia formy. Dodatkowo zastosowanie tutaj ma @ReadableName.

```
package pl.cezaryregec.zoo.actions.animals.get;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Getter;
import lombok.NoArgsConstructor;
import pl.cezaryregec.zoo.actions.ActionQuery;
import pl.cezaryregec.zoo.actions.animals.AnimalType;
import pl.cezaryregec.zoo.console.annotation.ReadableName;

@Builder
@AllArgsConstructor
@NoArgsConstructor
@Getter
public class GetAnimalsByTypeQuery implements ActionQuery {
    @ReadableName("Typ zwierzęcia")
    private AnimalType type;
}
```

Klasa GetAnimalsByTypeActionExecutor



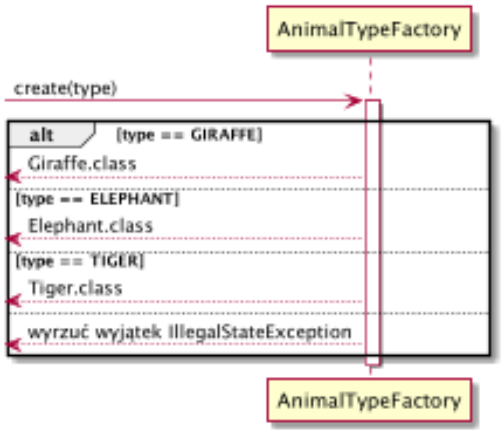
Ta klasa realizuje logikę operacji wyświetlenia zwierząt znajdujących się w repozytorium filtrując po typie (operacja GET_ANIMALS_BY_TYPE).



```java
package pl.cezaryregec.zoo.actions.animals.get;

import pl.cezaryregec.zoo.actions.ActionExecutor;
import pl.cezaryregec.zoo.actions.animals.AnimalTypeFactory;
import pl.cezaryregec.zoo.dto.result.ResultDto;
import pl.cezaryregec.zoo.repository.AnimalRepository;
import pl.cezaryregec.zoo.utils.ReflectionUtils;

public class GetAnimalsByTypeActionExecutor implements
ActionExecutor<GetAnimalsByTypeQuery> {

    @Override
    public ResultDto execute(GetAnimalsByTypeQuery query) {
        AnimalRepository repository =
ReflectionUtils.getInstance(AnimalRepository.class);
        Class<?> type = AnimalTypeFactory.create(query.getType());
        return ResultDto.of(repository.get(item -> item.getClass() == type));
    }
}
```

Klasa AnimalTypeFactory

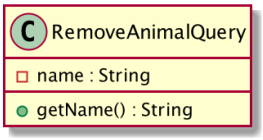
Ta klasa stanowi fabrykę klas dla danego rodzaju zwierzęcia




```java
package pl.cezaryregec.zoo.actions.animals;

import pl.cezaryregec.zoo.model.animal.Elephant;
import pl.cezaryregec.zoo.model.animal.Giraffe;
```

```
import pl.cezaryregec.zoo.model.animal.Tiger;

public class AnimalTypeFactory {
    public static Class<?> create(AnimalType type) {
        switch (type) {
            case GIRAFFE:
                return Giraffe.class;
            case ELEPHANT:
                return Elephant.class;
            case TIGER:
                return Tiger.class;
        }

        throw new IllegalStateException("Animal type " + type + " is not
supported");
    }
}
```

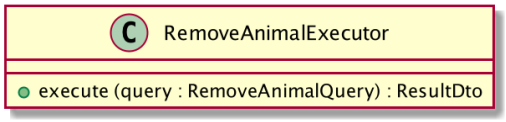
Klasa RemoveAnimalQuery



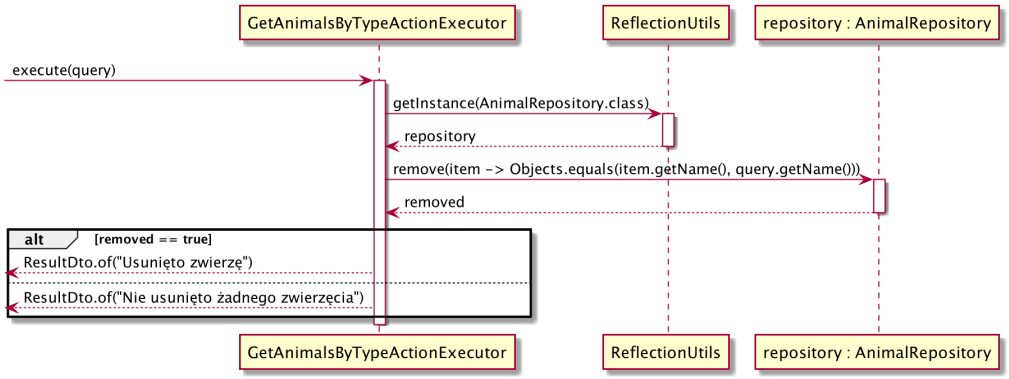
Klasa RemoveAnimalQuery jest klasą agregującą parametry operacji REMOVE_ANIMAL. Klasa wykorzystuje adnotacje biblioteki Lombok dla uproszczenia formy. Dodatkowo zastosowanie tutaj ma @ReadableName.

```
package pl.cezaryregec.zoo.actions.animals.remove;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Getter;
import lombok.NoArgsConstructor;
import pl.cezaryregec.zoo.actions.ActionQuery;
import pl.cezaryregec.zoo.console.annotation.ReadableName;

@Builder
@NoArgsConstructor
@AllArgsConstructor
@Getter
public class RemoveAnimalQuery implements ActionQuery {
    @ReadableName("Imię zwierzęcia")
    private String name;
}
```

Klasa RemoveAnimalActionExecutor

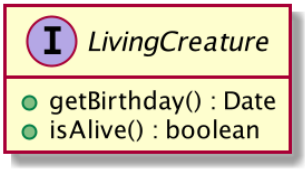


Ta klasa realizuje logikę operacji usuwania konkretnego zwierzęcia znajdującego się w repozytorium (operacja REMOVE_ANIMAL).




```
package pl.cezaryregec.zoo.actions.animals.remove;
import pl.cezaryregec.zoo.actions.ActionExecutor;
import pl.cezaryregec.zoo.dto.result.ResultDto;
import pl.cezaryregec.zoo.repository.AnimalRepository;
import pl.cezaryregec.zoo.utils.ReflectionUtils;

import java.util.Objects;

public class RemoveAnimalActionExecutor implements
ActionExecutor<RemoveAnimalQuery> {
    @Override
    public ResultDto execute(RemoveAnimalQuery query) {
        AnimalRepository repository =
ReflectionUtils.getInstance(AnimalRepository.class);
        boolean removed = repository.remove(item -> Objects.equals(item.getName(),
query.getName()));
        if (removed) {
            return ResultDto.of("Usunięto zwierzę");
        } else {
            return ResultDto.of("Nie usunięto żadnego zwierzęcia");
        }
    }
}
```

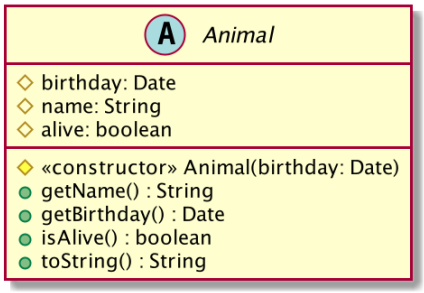
Interfejs LivingCreature



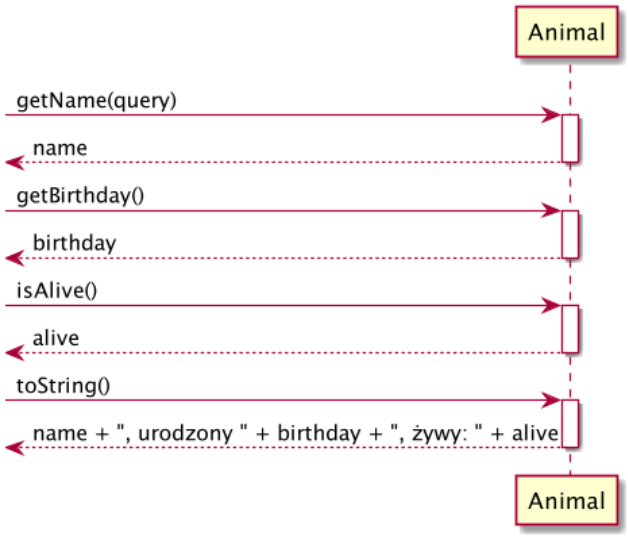
Interfejs stanowi wymaganie, które musi spełnić model reprezentujący żywe stworzenie.

```
package pl.cezaryregec.zoo.model;

import java.io.Serializable;
import java.time.LocalDate;

public interface LivingCreature extends Serializable {
    LocalDate getBirthday();
    boolean isAlive();
}
```

Abstrakcyjna klasa Animal

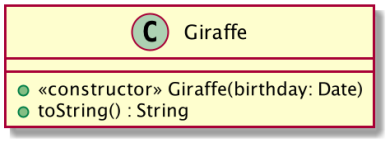


Klasa stanowi implementację elementów wspólnych dla modeli zwierząt, jednocześnie jest abstrakcyjną realizacją interfejsu LivingCreature.

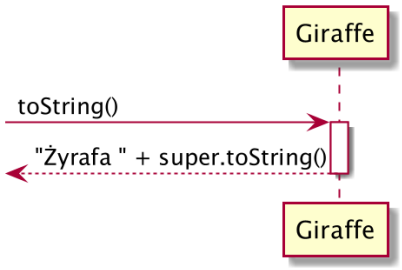


```java
package pl.cezaryregec.zoo.model.animal;

import pl.cezaryregec.zoo.model.LivingCreature;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.Calendar;
import java.util.Date;

public abstract class Animal implements LivingCreature {
    private static final long serialVersionUID = 5678831852485909559L;

    protected final String name;
    protected final LocalDate birthday;
    protected boolean alive;

    protected Animal(String name, LocalDate birthday) {
        this.name = name;
        this.birthday = birthday;
        this.alive = true;
    }

    public String getName() {
        return this.name;
    }

    public LocalDate getBirthday() {
        return this.birthday;
    }

    public boolean isAlive() {
        return this.alive;
    }

    @Override
    public boolean equals(Object obj) {
        if (obj instanceof Animal) {
            return this.name.equals(((Animal) obj).name);
        }
        return false;
    }

    @Override
    public String toString() {
        return name + ", urodzony " + birthday.format(DateTimeFormatter.ISO_DATE) +
", żywy: " + (alive ? "tak" : "nie");
    }
}
```

Klasa Giraffe

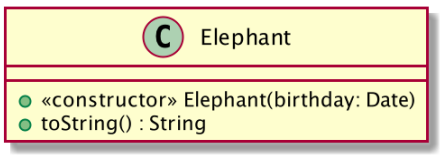


Klasa reprezentuje model żyrafy w systemie i dziedziczy po Animal.

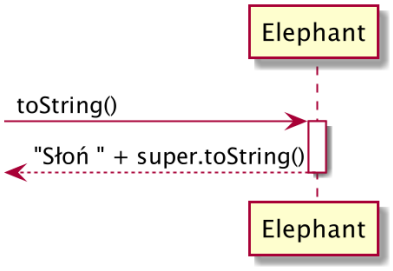



```
package pl.cezaryregec.zoo.model.animal;

import java.time.LocalDate;
import java.util.Calendar;

public class Giraffe extends Animal {
    private static final long serialVersionUID = 5966576888274965455L;

    public Giraffe(String name, LocalDate birthday) {
        super(name, birthday);
    }

    @Override
    public String toString() {
        return "Żyrafa " + super.toString();
    }
}
```

Klasa Elephant

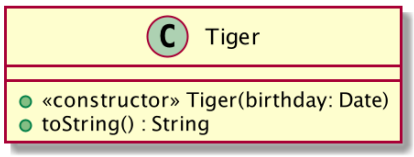


Klasa reprezentuje model słonia w systemie i dziedziczy po Animal.

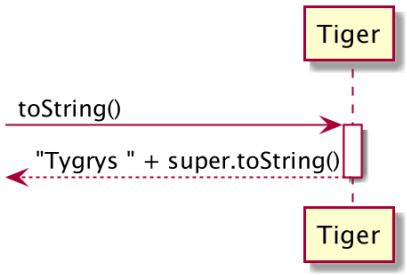


```java
package pl.cezaryregec.zoo.model.animal;

import java.time.LocalDate;
import java.util.Calendar;

public class Elephant extends Animal {
    private static final long serialVersionUID = -1980044156647194615L;

    public Elephant(String name, LocalDate birthday) {
        super(name, birthday);
    }

    @Override
    public String toString() {
        return "Słoń " + super.toString();
    }
}
```

Klasa Tiger



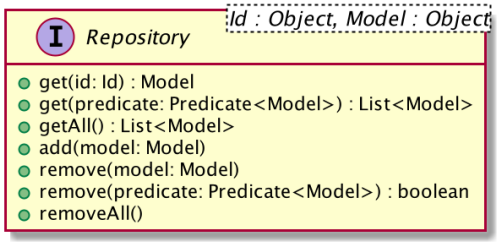
Klasa reprezentuje model tygrysa w systemie i dziedziczy po Animal.




```java
package pl.cezaryregec.zoo.model.animal;

import java.time.LocalDate;
import java.util.Calendar;

public class Tiger extends Animal {
    private static final long serialVersionUID = -6868309417903072158L;

    public Tiger(String name, LocalDate birthday) {
        super(name, birthday);
    }
```

```
    @Override
    public String toString() {
        return "Tygrys " + super.toString();
    }
}
```

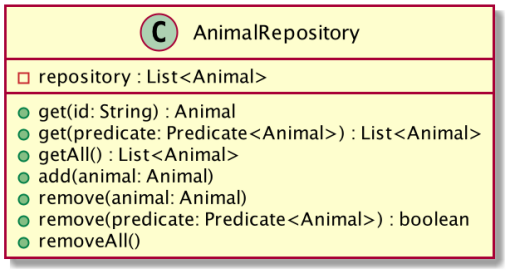
Interfejs Repository



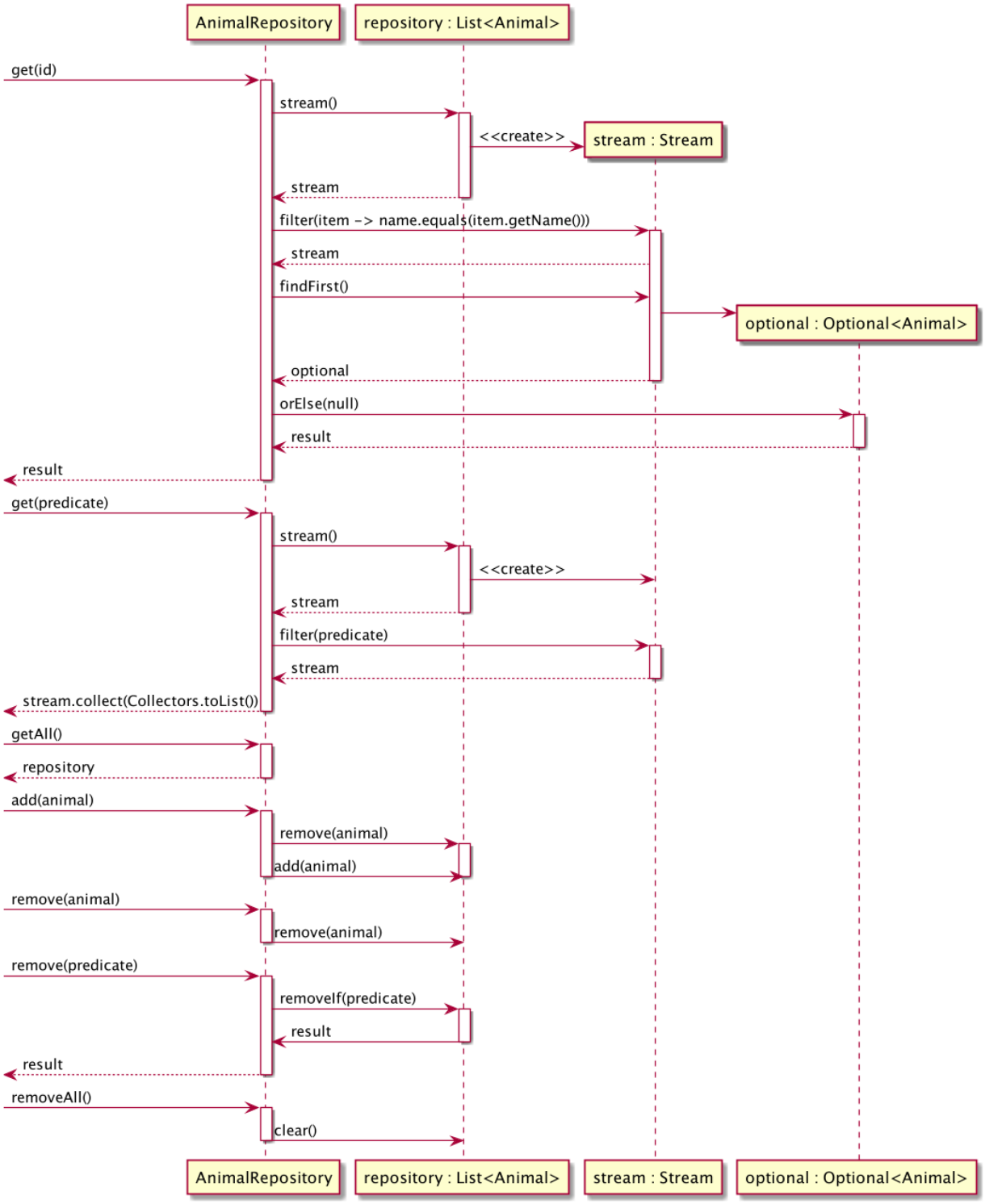
Interfejs stanowi wymaganie, jakie musi spełniać implementacja repozytorium danych.

```
package pl.cezaryregec.zoo.repository;

import java.io.Serializable;
import java.util.List;
import java.util.function.Predicate;

public interface Repository<Id, Model> extends Serializable {
    Model get(Id id);
    List<Model> get(Predicate<Model> predicate);
    List<Model> getAll();
    void add(Model model);
    void remove(Model model);
    boolean remove(Predicate<Model> predicate);
    void removeAll();
}
```

Klasa AnimalRepository



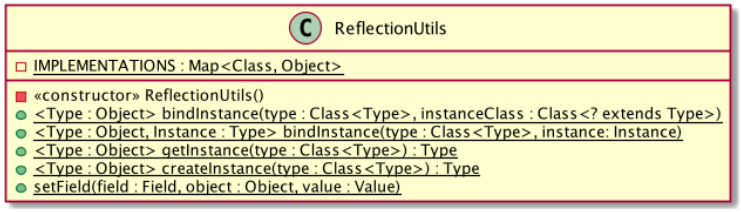
Klasa ta jest implementacją repozytorium zwierząt (typu Animal).



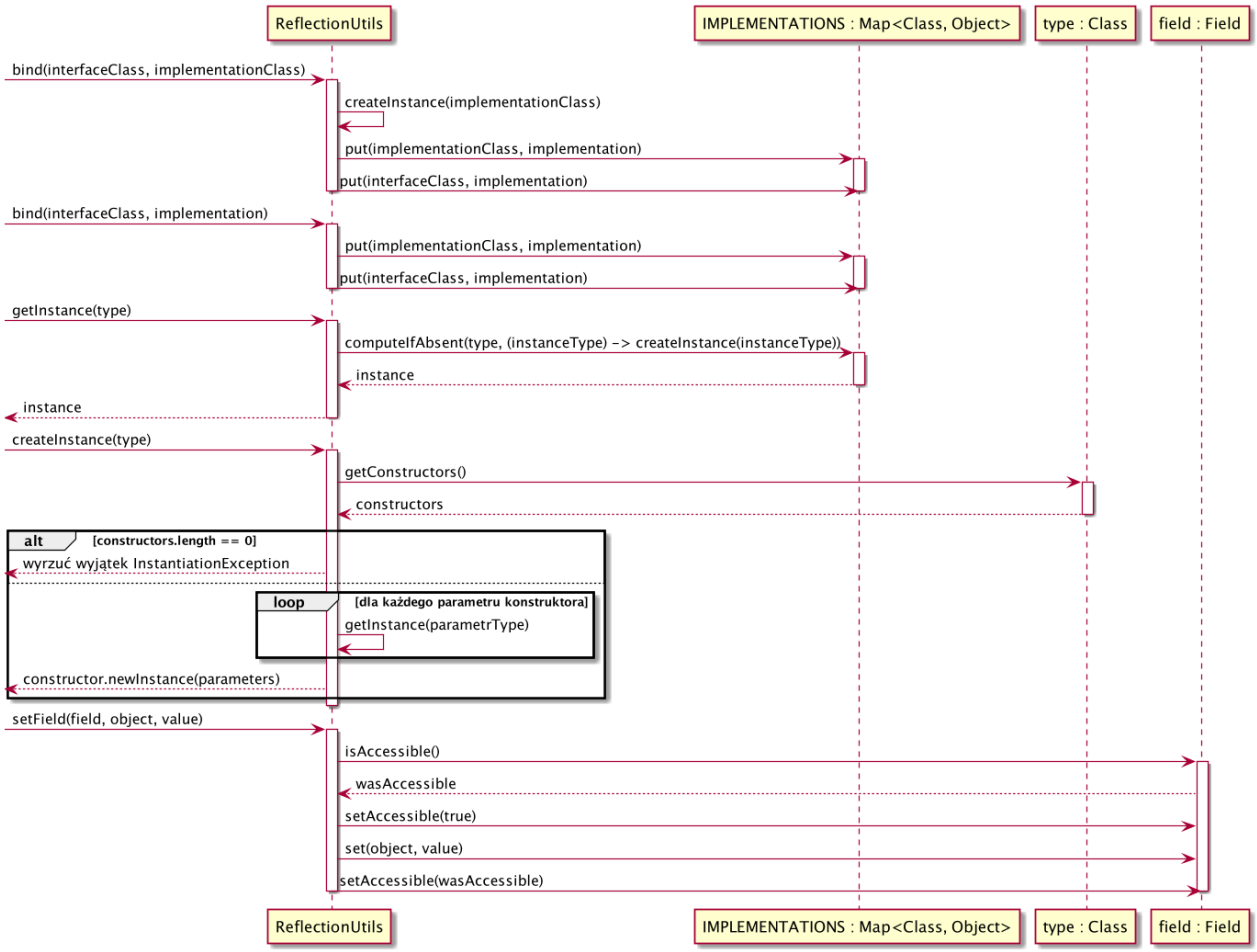

```java
package pl.cezaryregec.zoo.repository;

import pl.cezaryregec.zoo.model.animal.Animal;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.function.Predicate;
```

```
import java.util.stream.Collectors;

public class AnimalRepository implements Repository<String, Animal> {
    private static final long serialVersionUID = -3004718513968458912L;

    private final List<Animal> repository = Collections.synchronizedList(new
ArrayList<>());

    @Override
    public synchronized Animal get(String name) {
        return repository.stream()
                .filter(item -> name.equals(item.getName()))
                .findFirst()
                .orElse(null);
    }

    @Override
    public synchronized List<Animal> get(Predicate<Animal> predicate) {
        return repository.stream()
                .filter(predicate)
                .collect(Collectors.toList());
    }

    @Override
    public synchronized List<Animal> getAll() {
        return new ArrayList<>(repository);
    }

    @Override
    public synchronized void add(Animal animal) {
        repository.remove(animal);
        repository.add(animal);
    }

    @Override
    public synchronized void remove(Animal animal) {
        repository.remove(animal);
    }

    @Override
    public synchronized boolean remove(Predicate<Animal> predicate) {
        return repository.removeIf(predicate);
    }

    @Override
    public synchronized void removeAll() {
        repository.clear();
    }
}
```

Klasa ReflectionUtils



Ta klasa jest klasą pomocniczą przy refleksji (ustawianiu wartości pól ActionQuery oraz tworzeniu instancji) i zarządzania podpiętymi instancjami przy wstrzykiwaniu zależności (oraz wstrzykiwaniu ich przez konstruktor, jeśli jest możliwość).




```
package pl.cezaryregec.zoo.utils;

import java.lang.reflect.Constructor;
import java.lang.reflect.Field;
import java.lang.reflect.InvocationTargetException;
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

public class ReflectionUtils {

    private static final Map<Class<?>, Object> IMPLEMENTATIONS = new
ConcurrentHashMap<>();

    private ReflectionUtils() {}

    public static <T> void bind (Class<T> type, Class<? extends T>
implementationClass) {
        T instance = createInstance(implementationClass);
        IMPLEMENTATIONS.put(implementationClass, instance);
        IMPLEMENTATIONS.put(type, instance);
    }

    public static <T, I extends T> void bind(Class<I> type, I implementation) {
        IMPLEMENTATIONS.put(implementation.getClass(), implementation);
        IMPLEMENTATIONS.put(type, implementation);
    }

    public static <T> T getInstance(Class<T> type) {
        return (T) IMPLEMENTATIONS.computeIfAbsent(type, (instanceType) ->
createInstance(instanceType));
    }

    public static <T> T createInstance(Class<T> type) {
        try {
            Constructor<?>[] declaredConstructors = type.getDeclaredConstructors();
            if (declaredConstructors.length > 1) {
                throw new InstantiationException("More than 1 constructor is
present, cannot instantiate " + type.getCanonicalName());
            }
            if (declaredConstructors.length == 0) {
                throw new InstantiationException("Cannot make an instance of an
interface: " + type.getCanonicalName());
            }

            Constructor constructor = declaredConstructors[0];
            Class[] parameterTypes = constructor.getParameterTypes();
            List<Object> parameters = new ArrayList<>();
            for (Class<?> parameter : parameterTypes) {
                parameters.add(getInstance(parameter));
            }

            return (T) constructor.newInstance(parameters.toArray());
        } catch (InstantiationException | IllegalAccessException |
InvocationTargetException e) {
```

```
            throw new RuntimeException(e);
        }
    }

    public static void setField(Field field, Object object, Object value) {
        boolean wasAccessible = field.isAccessible();
        try {
            field.setAccessible(true);
            field.set(object, value);
            field.setAccessible(wasAccessible);
        } catch (IllegalAccessException e) {
            throw new RuntimeException(e);
        }
    }
}
```

Przykład działania programu

```
[GET_MENU] Menu dostępnych opcji
[ADD_ANIMAL] Dodaj zwierzę
[GET_ANIMALS] Wyświetl zwierzęta w ZOO
[GET_ANIMALS_BY_TYPE] Wyświetl zwierzęta w ZOO filtrując po typie
[REMOVE_ANIMAL] Usuń zwierzę
[SHUTDOWN] Zakończ program

Opcja: ADD_ANIMAL
Rodzaj zwierzęcia (GIRAFFE, ELEPHANT, TIGER): TIGER
Imię zwierzęcia: Andrzej
Rok urodzenia: 2010
Miesiąc urodzenia: 02
Dzień urodzenia: 11
Tygrys Andrzej, urodzony 2010-02-11, żywy: tak

Opcja: ADD_ANIMAL
Rodzaj zwierzęcia (GIRAFFE, ELEPHANT, TIGER): GIRAFFE
Imię zwierzęcia: Izabela
Rok urodzenia: 2011
Miesiąc urodzenia: 09
Dzień urodzenia: 26
Żyrafa Izabela, urodzony 2011-09-26, żywy: tak

Opcja: GET_ANIMALS
Tygrys Andrzej, urodzony 2010-02-11, żywy: tak
Żyrafa Izabela, urodzony 2011-09-26, żywy: tak

Opcja: GET_ANIMALS_BY_TYPE
Typ zwierzęcia (GIRAFFE, ELEPHANT, TIGER): TIGER
Tygrys Andrzej, urodzony 2010-02-11, żywy: tak

Opcja: REMOVE_ANIMAL
Imię zwierzęcia: Andrzej
Usunięto zwierzę

Opcja: GET_ANIMALS
Żyrafa Izabela, urodzony 2011-09-26, żywy: tak

Opcja: SHUTDOWN
```

Testy

Poniższe raporty z testów BDD sprawdzają, czy aplikacja działa zgodnie z wymaganiami określonymi przez przypadki użycia. Testy zostały wykonane z użyciem bibliotek JUnit 4.12, JGiven 0.17.1, Mockito 2.28.2, AssertJ 3.11 oraz JUnitParams 1.1.1.

Test Class: pl.cezaryregec.zoo.scenario.menu.MenuTest

Menu

Kiedy wybieram opcję GET_MENU
Wtedy wynik zawiera tylko

wartości
[GET_MENU] Menu dostępnych opcji
[ADD_ANIMAL] Dodaj zwierzę
[GET_ANIMALS] Wyświetl zwierzęta w ZOO
[GET_ANIMALS_BY_TYPE] Wyświetl zwierzęta w ZOO filtrując po typie
[REMOVE_ANIMAL] Usuń zwierzę
[SHUTDOWN] Zakończ program

Test Class: pl.cezaryregec.zoo.scenario.animals.AnimalsTest

Dodanie zwierzęcia

Case 1: typ = GIRAFFE, imię = Cezary, rokUrodzenia = 2010, miesiącUrodzenia = 2, dzieńUrodzenia = 11, wynik = "Żyrafa Cezary, urodzony 2010-02-11, żywy: tak"

Zakładając że repozytorium jest puste
Kiedy wybieram opcję ADD_ANIMAL z parametrami

Rodzaj zwierzęcia	Imię zwierzęcia	Rok urodzenia	Miesiąc urodzenia	Dzień urodzenia
GIRAFFE	Cezary	2010	2	11

Wtedy wynik zawiera tylko "Żyrafa Cezary, urodzony 2010-02-11, żywy: tak"
Oraz repozytorium zawiera zwierzę "Cezary"

Case 2: typ = ELEPHANT, imię = Krzysztof, rokUrodzenia = 1999, miesiącUrodzenia = 8, dzieńUrodzenia = 9, wynik = "Słoń Krzysztof, urodzony 1999-08-09, żywy: tak"

Zakładając że repozytorium jest puste
Kiedy wybieram opcję ADD_ANIMAL z parametrami

Rodzaj zwierzęcia	Imię zwierzęcia	Rok urodzenia	Miesiąc urodzenia	Dzień urodzenia
ELEPHANT	Krzysztof	1999	8	9

Wtedy wynik zawiera tylko "Słoń Krzysztof, urodzony 1999-08-09, żywy: tak"
Oraz repozytorium zawiera zwierzę "Krzysztof"

Nadpisanie zwierzęcia

Zakładając że w repozytorium znajduje się Żyrafa Andrzej, urodzony 2000-01-01, żywy: tak
Kiedy wybieram opcję ADD_ANIMAL z parametrami

Rodzaj zwierzęcia	Imię zwierzęcia	Rok urodzenia	Miesiąc urodzenia	Dzień urodzenia
TIGER	Andrzej	2010	9	26

Wtedy wynik zawiera tylko "Tygrys Andrzej, urodzony 2010-09-26, żywy: tak"
Oraz repozytorium zawiera zwierzę "Andrzej"
Oraz Andrzej tygrysem

Wyświetlenie wszystkich zwierząt

Zakładając że w repozytorium znajduje się Słoń Tadeusz, urodzony 2000-01-01, żywy: tak
Oraz w repozytorium znajduje się Tygrys Ryszard, urodzony 2000-01-01, żywy: tak
Kiedy wybieram opcję GET_ANIMALS
Wtedy wynik zawiera tylko

wartości
Słoń Tadeusz, urodzony 2000-01-01, żywy: tak
Tygrys Ryszard, urodzony 2000-01-01, żywy: tak

Wyświetlenie zwierząt po typie

Zakładając że w repozytorium znajduje się Słoń Janusz, urodzony 2000-01-01, żywy: tak
Oraz w repozytorium znajduje się Żyrafa Anita, urodzony 2000-01-01, żywy: tak
Oraz w repozytorium znajduje się Tygrys Piotr, urodzony 2000-01-01, żywy: tak
Kiedy wybieram opcję GET_ANIMALS_BY_TYPE z parametrami

Typ zwierzęcia
GIRAFFE

Wtedy wynik zawiera tylko "Żyrafa Anita, urodzony 2000-01-01, żywy: tak"

Udane usunięcie zwierzęcia

Case 1: typ = TIGER, imię = Grzegorz

Zakładając że w repozytorium znajduje się Tygrys Grzegorz, urodzony 2000-01-01, żywy: tak
Kiedy wybieram opcję REMOVE_ANIMAL z parametrami

Imię zwierzęcia
Grzegorz

Wtedy wynik zawiera tylko "Usunięto zwierzę"
Oraz repozytorium nie zawiera zwierzęcia "Grzegorz"

Case 2: typ = GIRAFFE, imię = Alicja

Zakładając że w repozytorium znajduje się Żyrafa Alicja, urodzony 2000-01-01, żywy: tak
Kiedy wybieram opcję REMOVE_ANIMAL z parametrami

Imię zwierzęcia
Alicja

Wtedy wynik zawiera tylko "Usunięto zwierzę"
Oraz repozytorium nie zawiera zwierzęcia "Alicja"

Nieudane usunięcie zwierzęcia

Case 1: imię = Grzegorz

Zakładając że repozytorium jest puste
Kiedy wybieram opcję REMOVE_ANIMAL z parametrami

Imię zwierzęcia
Grzegorz

Wtedy wynik zawiera tylko "Nie usunięto żadnego zwierzęcia"

Case 2: imię = Alicja

Zakładając że repozytorium jest puste
Kiedy wybieram opcję REMOVE_ANIMAL z parametrami

Imię zwierzęcia
Alicja

Wtedy wynik zawiera tylko "Nie usunięto żadnego zwierzęcia”

Test Class: pl.cezaryregec.zoo.scenario.shutdown.ShutdownTest

Wyłączenie aplikacji

Kiedy wybieram opcję SHUTDOWN
Wtedy nastąpił wyjątek zakończenia aplikacji

Kod źródłowy testów

Klasa pl.cezaryregec.zoo.scenario.menu.MenuTest

```
package pl.cezaryregec.zoo.scenario.menu;

import org.junit.Test;
import pl.cezaryregec.zoo.scenario.PolishScenarioTest;
import pl.cezaryregec.zoo.actions.ZooActionIndex;
import pl.cezaryregec.zoo.actions.menu.GetMenuQuery;
import pl.cezaryregec.zoo.stages.given.AnimalRepositoryState;
import pl.cezaryregec.zoo.stages.then.ActionOutcome;
import pl.cezaryregec.zoo.stages.when.ApplicationAction;

public class MenuTest extends PolishScenarioTest<AnimalRepositoryState, ApplicationAction,
ActionOutcome> {

    @Test
    public void menu() {
        kiedy().wybieramOpcję(ZooActionIndex.GET_MENU, GetMenuQuery.class);
        wtedy().wynikZawieraTylko(
                "[GET_MENU] Menu dostępnych opcji",
                "[ADD_ANIMAL] Dodaj zwierzę",
                "[GET_ANIMALS] Wyświetl zwierzęta w ZOO",
                "[GET_ANIMALS_BY_TYPE] Wyświetl zwierzęta w ZOO filtrując po typie",
                "[REMOVE_ANIMAL] Usuń zwierzę",
                "[SHUTDOWN] Zakończ program"
        );
    }
}
```

Klasa pl.cezaryregec.zoo.scenario.animals.AnimalsTest

```
package pl.cezaryregec.zoo.scenario.animals;

import com.tngtech.jgiven.annotation.Quoted;
import com.tngtech.jgiven.annotation.ScenarioStage;
import junitparams.JUnitParamsRunner;
import junitparams.Parameters;
import org.junit.Test;
import org.junit.runner.RunWith;
import pl.cezaryregec.zoo.actions.ZooActionIndex;
import pl.cezaryregec.zoo.actions.animals.AnimalType;
import pl.cezaryregec.zoo.actions.animals.add.AddAnimalQuery;
import pl.cezaryregec.zoo.actions.animals.get.GetAnimalsByTypeQuery;
import pl.cezaryregec.zoo.actions.animals.get.GetAnimalsQuery;
import pl.cezaryregec.zoo.actions.animals.remove.RemoveAnimalQuery;
import pl.cezaryregec.zoo.scenario.PolishScenarioTest;
import pl.cezaryregec.zoo.stages.given.AnimalPrototypes;
import pl.cezaryregec.zoo.stages.given.AnimalRepositoryState;
import pl.cezaryregec.zoo.stages.then.ActionOutcome;
import pl.cezaryregec.zoo.stages.then.AnimalRepositoryOutcome;
import pl.cezaryregec.zoo.stages.when.ApplicationAction;
```

```
@RunWith(JUnitParamsRunner.class)
public class AnimalsTest extends PolishScenarioTest<AnimalRepositoryState, ApplicationAction,
ActionOutcome> {

    @ScenarioStage
    AnimalRepositoryOutcome repositoryOutcome;

    @Test
    @Parameters({
            "GIRAFFE, Cezary, 2010, 2, 11, Żyrafa Cezary\\, urodzony 2010-02-11\\, żywy:
tak",
            "ELEPHANT, Krzysztof, 1999, 8, 9, Słoń Krzysztof\\, urodzony 1999-08-09\\, żywy:
tak"
    })
    public void dodanieZwierzęcia(AnimalType typ, String imię, Integer rokUrodzenia, Integer
miesiącUrodzenia, Integer dzieńUrodzenia, @Quoted String wynik) {
        zakładającŻe().repozytoriumJestPuste();
        kiedy().wybieramOpcję$ZParametrami(ZooActionIndex.ADD_ANIMAL,
AddAnimalQuery.builder()
                .type(typ)
                .name(imię)
                .yearOfBirth(rokUrodzenia)
                .monthOfBirth(miesiącUrodzenia)
                .dayOfBirth(dzieńUrodzenia)
                .build());
        wtedy().wynikZawieraTylko(wynik);
        repositoryOutcome
                .oraz().repozytoriumZawieraZwierzę(imię);
    }

    @Test
    public void nadpisanieZwierzęcia() {
        zakładającŻe().wRepozytoriumZnajdujeSię(AnimalPrototypes.createGiraffe("Andrzej"));
        kiedy().wybieramOpcję$ZParametrami(ZooActionIndex.ADD_ANIMAL,
AddAnimalQuery.builder()
                .type(AnimalType.TIGER)
                .name("Andrzej")
                .yearOfBirth(2010)
                .monthOfBirth(9)
                .dayOfBirth(26)
                .build());
        wtedy().wynikZawieraTylko("Tygrys Andrzej, urodzony 2010-09-26, żywy: tak");
        repositoryOutcome
                .oraz().repozytoriumZawieraZwierzę("Andrzej")
                .oraz().$jestTygrysem("Andrzej");
    }

    @Test
    @Parameters({
            "TIGER, Grzegorz",
            "GIRAFFE, Alicja"
    })
    public void udaneUsunięcieZwierzęcia(AnimalType typ, String imię) {
        zakładającŻe().wRepozytoriumZnajdujeSię(AnimalPrototypes.create(typ, imię));
        kiedy().wybieramOpcję$ZParametrami(ZooActionIndex.REMOVE_ANIMAL,
RemoveAnimalQuery.builder().name(imię).build());
```

```
        wtedy().wynikZawieraTylko("Usunięto zwierzę");
        repositoryOutcome
                .oraz().repozytoriumNieZawieraZwierzęcia(imię);
    }

    @Test
    @Parameters({
            "Grzegorz",
            "Alicja"
    })
    public void nieudaneUsunięcieZwierzęcia(String imię) {
        zakładającŻe().repozytoriumJestPuste();
        kiedy().wybieramOpcję$ZParametrami(ZooActionIndex.REMOVE_ANIMAL,
RemoveAnimalQuery.builder().name(imię).build());
        wtedy().wynikZawieraTylko("Nie usunięto żadnego zwierzęcia");
    }

    @Test
    public void wyświetlenieWszystkichZwierząt() {
        zakładającŻe().wRepozytoriumZnajdujeSię(AnimalPrototypes.createElephant("Tadeusz"))
                .oraz().wRepozytoriumZnajdujeSię(AnimalPrototypes.createTiger("Ryszard"));
        kiedy().wybieramOpcję(ZooActionIndex.GET_ANIMALS, GetAnimalsQuery.class);
        wtedy().wynikZawieraTylko(
                "Słoń Tadeusz, urodzony 2000-01-01, żywy: tak",
                "Tygrys Ryszard, urodzony 2000-01-01, żywy: tak"
        );
    }

    @Test
    public void wyświetlenieZwierzątPoTypie() {
        zakładającŻe().wRepozytoriumZnajdujeSię(AnimalPrototypes.createElephant("Janusz"))
                .oraz().wRepozytoriumZnajdujeSię(AnimalPrototypes.createGiraffe("Anita"))
                .oraz().wRepozytoriumZnajdujeSię(AnimalPrototypes.createTiger("Piotr"));
        kiedy().wybieramOpcję$ZParametrami(ZooActionIndex.GET_ANIMALS_BY_TYPE,
GetAnimalsByTypeQuery.builder()
                        .type(AnimalType.GIRAFFE)
                        .build()
        );
        wtedy().wynikZawieraTylko("Żyrafa Anita, urodzony 2000-01-01, żywy: tak");
    }
}
```

Klasa pl.cezaryregec.zoo.scenario.shutdown.ShutdownTest

```
package pl.cezaryregec.zoo.scenario.shutdown;

import org.junit.Test;
import pl.cezaryregec.zoo.actions.ZooActionIndex;
import pl.cezaryregec.zoo.actions.shutdown.ShutdownQuery;
import pl.cezaryregec.zoo.scenario.PolishScenarioTest;
import pl.cezaryregec.zoo.stages.given.EmptyState;
import pl.cezaryregec.zoo.stages.then.ActionOutcome;
import pl.cezaryregec.zoo.stages.when.ApplicationAction;
```

```
public class ShutdownTest extends PolishScenarioTest<EmptyState, ApplicationAction,
ActionOutcome> {
    @Test
    public void wyłączenieAplikacji() {
        kiedy().wybieramOpcję(ZooActionIndex.SHUTDOWN, ShutdownQuery.class);
        wtedy().nastąpiłWyjątekZakończeniaAplikacji();
    }
}
```

Klasa pl.cezaryregec.zoo.scenario.PolishScenarioTest

```
package pl.cezaryregec.zoo.scenario;

import com.tngtech.jgiven.junit.ScenarioTest;

public class PolishScenarioTest<ZAKLADAJAC, KIEDY, WTEDY> extends ScenarioTest<ZAKLADAJAC,
KIEDY, WTEDY> {
    public ZAKLADAJAC zakładającŻe() {
        return getScenario().given("Zakładając że");
    }

    public KIEDY kiedy() {
        return getScenario().when("Kiedy");
    }

    public WTEDY wtedy() {
        return getScenario().then("Wtedy");
    }
}
```

Klasa pl.cezaryregec.zoo.stages.given.AnimalRepositoryState

```
package pl.cezaryregec.zoo.stages.given;

import com.tngtech.jgiven.annotation.BeforeStage;
import pl.cezaryregec.zoo.model.animal.Animal;
import pl.cezaryregec.zoo.repository.AnimalRepository;
import pl.cezaryregec.zoo.stages.PolishStage;
import pl.cezaryregec.zoo.utils.ReflectionUtils;

public class AnimalRepositoryState extends PolishStage<AnimalRepositoryState> {
    @BeforeStage
    public void prepareRepository() {
        AnimalRepository repository = ReflectionUtils.getInstance(AnimalRepository.class);
        repository.removeAll();
    }

    public AnimalRepositoryState wRepozytoriumZnajdujeSię(Animal animal) {
        AnimalRepository repository = ReflectionUtils.getInstance(AnimalRepository.class);
        repository.add(animal);
        return self();
    }

    public AnimalRepositoryState repozytoriumJestPuste() {
```

```
        AnimalRepository repository = ReflectionUtils.getInstance(AnimalRepository.class);
        repository.removeAll();
        return self();
    }
}
```

Klasa pl.cezaryregec.zoo.stages.given.EmptyState

```
package pl.cezaryregec.zoo.stages.given;

import pl.cezaryregec.zoo.stages.PolishStage;

public class EmptyState extends PolishStage<EmptyState> {}
```

Klasa pl.cezaryregec.zoo.stages.given.AnimalPrototypes

```
package pl.cezaryregec.zoo.stages.given;

import pl.cezaryregec.zoo.actions.animals.AnimalType;
import pl.cezaryregec.zoo.model.animal.Animal;
import pl.cezaryregec.zoo.model.animal.Elephant;
import pl.cezaryregec.zoo.model.animal.Giraffe;
import pl.cezaryregec.zoo.model.animal.Tiger;

import java.time.LocalDate;

public class AnimalPrototypes {
    public static final LocalDate BIRTHDAY = LocalDate.of(2000, 1, 1);

    public static Animal create(AnimalType type, String name) {
        switch (type) {
            case GIRAFFE:
                return createGiraffe(name);
            case ELEPHANT:
                return createElephant(name);
            case TIGER:
                return createTiger(name);
        }
        throw new IllegalStateException("Type not supported: " + type);
    }

    public static Giraffe createGiraffe(String name) {
        return new Giraffe(name, BIRTHDAY);
    }

    public static Elephant createElephant(String name) {
        return new Elephant(name, BIRTHDAY);
    }

    public static Tiger createTiger(String name) {
        return new Tiger(name, BIRTHDAY);
    }
}
```

Klasa pl.cezaryregec.zoo.stages.when.ApplicationAction

```
package pl.cezaryregec.zoo.stages.when;

import com.tngtech.jgiven.annotation.ExpectedScenarioState;
import com.tngtech.jgiven.annotation.Hidden;
import com.tngtech.jgiven.annotation.ProvidedScenarioState;
import com.tngtech.jgiven.annotation.Table;
import org.mockito.Mockito;
import pl.cezaryregec.zoo.actions.ActionExecutor;
import pl.cezaryregec.zoo.actions.ActionQuery;
import pl.cezaryregec.zoo.actions.ZooActionFactory;
import pl.cezaryregec.zoo.actions.ZooActionIndex;
import pl.cezaryregec.zoo.actions.animals.add.AddAnimalQuery;
import pl.cezaryregec.zoo.dto.result.ResultDto;
import pl.cezaryregec.zoo.formatter.ActionQueryTableFormatterFactory;
import pl.cezaryregec.zoo.stages.PolishStage;
import pl.cezaryregec.zoo.utils.ReflectionUtils;

public class ApplicationAction extends PolishStage<ApplicationAction> {
    @ProvidedScenarioState
    private ResultDto resultDto;

    @ProvidedScenarioState
    private Exception exception;

    public ApplicationAction wybieramOpcję(ZooActionIndex opcja, @Hidden Class<? extends
ActionQuery> queryClass) {
        try {
            ZooActionFactory actionFactory =
ReflectionUtils.createInstance(ZooActionFactory.class);
            ActionExecutor actionExecutor = actionFactory.create(opcja);
            ActionQuery query = Mockito.mock(queryClass);
            resultDto = actionExecutor.execute(query);
        } catch (Exception ex) {
            exception = ex;
        }
        return self();
    }

    public ApplicationAction wybieramOpcję$ZParametrami(ZooActionIndex opcja,
@Table(rowFormatter = ActionQueryTableFormatterFactory.class) ActionQuery parametry) {
        try {
            ZooActionFactory actionFactory =
ReflectionUtils.createInstance(ZooActionFactory.class);
            ActionExecutor actionExecutor = actionFactory.create(opcja);
            resultDto = actionExecutor.execute(parametry);
        } catch (Exception ex) {
            exception = ex;
        }
        return self();
    }
}
```

Klasa pl.cezaryregec.zoo.stages.then.ActionOutcome

```
package pl.cezaryregec.zoo.stages.then;

import com.tngtech.jgiven.annotation.ExpectedScenarioState;
import com.tngtech.jgiven.annotation.Quoted;
import com.tngtech.jgiven.annotation.Table;
import org.assertj.core.api.Assertions;
import pl.cezaryregec.zoo.dto.result.ResultDto;
import pl.cezaryregec.zoo.exception.ShutdownRequestException;
import pl.cezaryregec.zoo.stages.PolishStage;

import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;

public class ActionOutcome extends PolishStage<ActionOutcome> {
    @ExpectedScenarioState
    private ResultDto resultDto;

    @ExpectedScenarioState
    private Exception exception;

    public ActionOutcome wynikZawieraTylko(@Quoted String wartość) {
        Assertions.assertThat(resultDto.toString()).isEqualTo(wartość);
        return self();
    }

    public ActionOutcome wynikZawieraTylko(@Table(objectFormatting =
Table.ObjectFormatting.PLAIN) String... wartości) {
        String[] values = resultDto.toString().split("\n");
        List<String> expected = Stream.of(wartości).collect(Collectors.toList());
        Assertions.assertThat(values).hasSameElementsAs(expected);
        return self();
    }

    public ActionOutcome nastąpiłWyjątekZakończeniaAplikacji() {
        Assertions.assertThat(exception)
                .isNotNull()
                .isInstanceOf(ShutdownRequestException.class);

        return self();
    }
}
```

Klasa pl.cezaryregec.zoo.stages.then.AnimalRepositoryOutcome

```
package pl.cezaryregec.zoo.stages.then;

import com.tngtech.jgiven.annotation.Quoted;
import org.assertj.core.api.Assertions;
import pl.cezaryregec.zoo.model.animal.Animal;
import pl.cezaryregec.zoo.model.animal.Tiger;
```

```
import pl.cezaryregec.zoo.repository.AnimalRepository;
import pl.cezaryregec.zoo.stages.PolishStage;
import pl.cezaryregec.zoo.utils.ReflectionUtils;

public class AnimalRepositoryOutcome extends PolishStage<AnimalRepositoryOutcome> {
    public AnimalRepositoryOutcome repozytoriumZawieraZwierzę(@Quoted String imię) {
        AnimalRepository animalRepository =
ReflectionUtils.getInstance(AnimalRepository.class);
        Animal animal = animalRepository.get(imię);
        Assertions.assertThat(animal).isNotNull();
        return self();
    }

    public AnimalRepositoryOutcome repozytoriumNieZawieraZwierzęcia(@Quoted String imię) {
        AnimalRepository animalRepository =
ReflectionUtils.getInstance(AnimalRepository.class);
        Animal animal = animalRepository.get(imię);
        Assertions.assertThat(animal).isNull();
        return self();
    }

    public AnimalRepositoryOutcome $jestTygrysem(String imię) {
        AnimalRepository animalRepository =
ReflectionUtils.getInstance(AnimalRepository.class);
        Animal animal = animalRepository.get(imię);
        Assertions.assertThat(animal).isInstanceOf(Tiger.class);
        return self();
    }
}
```

Klasa pl.cezaryregec.zoo.stages.PolishStage

```
package pl.cezaryregec.zoo.stages;

import com.tngtech.jgiven.Stage;
import com.tngtech.jgiven.annotation.IntroWord;

public class PolishStage<SELF extends PolishStage<?>> extends Stage<SELF> {
    @IntroWord
    public SELF zakładającŻe() {
        return super.given();
    }

    @IntroWord
    public SELF kiedy() {
        return super.when();
    }

    @IntroWord
    public SELF wtedy() {
        return super.then();
    }

    @IntroWord
```

```
    public SELF oraz() {
        return super.and();
    }

    @IntroWord
    public SELF ale() {
        return super.but();
    }
}
```

Klasa pl.cezaryregec.zoo.formatter.ActionQueryTableFormatterFactory

```
package pl.cezaryregec.zoo.formatter;

import com.tngtech.jgiven.annotation.Table;
import com.tngtech.jgiven.config.FormatterConfiguration;
import com.tngtech.jgiven.format.ObjectFormatter;
import com.tngtech.jgiven.format.table.RowFormatter;
import com.tngtech.jgiven.format.table.RowFormatterFactory;

import java.lang.annotation.Annotation;

public class ActionQueryTableFormatterFactory implements RowFormatterFactory {

    @Override
    public RowFormatter create(Class<?> aClass, String s, Table table, Annotation[]
annotations, FormatterConfiguration formatterConfiguration, ObjectFormatter<?>
objectFormatter) {
        return new ActionQueryRowFormatter(aClass, s, table, annotations);
    }
}
```

Klasa pl.cezaryregec.zoo.formatter.ActionQueryRowFormatter

```
package pl.cezaryregec.zoo.formatter;

import com.google.common.collect.Sets;
import com.tngtech.jgiven.annotation.Table;
import com.tngtech.jgiven.format.table.FieldBasedRowFormatter;
import com.tngtech.jgiven.impl.util.ReflectionUtil;
import pl.cezaryregec.zoo.console.annotation.ReadableName;

import java.lang.annotation.Annotation;
import java.lang.reflect.Field;
import java.util.List;
import java.util.Set;
import java.util.stream.Collectors;
import java.util.stream.StreamSupport;

public class ActionQueryRowFormatter extends FieldBasedRowFormatter {

    private List<Field> fields;

    public ActionQueryRowFormatter(Class<?> type, String parameterName, Table
tableAnnotation, Annotation[] annotations) {
```

```
        super(type, parameterName, tableAnnotation, annotations);
        this.fields = getFields(tableAnnotation, type);
    }

    @Override
    public List<String> header() {
        return getFieldNames(this.fields);
    }

    private List<String> getFieldNames(Iterable<Field> fields) {
        return StreamSupport.stream(fields.spliterator(), false)
                .map(this::mapName)
                .collect(Collectors.toList());
    }

    private String mapName(Field field) {
        if (field.isAnnotationPresent(ReadableName.class)) {
            return field.getAnnotation(ReadableName.class).value();
        }

        return field.getName().replace('_', ' ');
    }

    private static List<Field> getFields(Table tableAnnotation, Class<?> type) {
        final Set<String> includeFields = Sets.newHashSet(tableAnnotation.includeFields());
        final Set<String> excludeFields = Sets.newHashSet(tableAnnotation.excludeFields());
        return ReflectionUtil.getAllNonStaticFields(type).stream().filter(input -> {
            String name = input.getName();
            if (!includeFields.isEmpty()) {
                return includeFields.contains(name);
            } else {
                return !excludeFields.contains(name);
            }
        }).collect(Collectors.toList());
    }
}
```

Załącznik - pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/
maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>pl.cezaryregec</groupId>
    <artifactId>zoo</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    </properties>

    <repositories>
        <repository>
            <id>central</id>
            <name>bintray</name>
            <url>http://jcenter.bintray.com</url>
        </repository>
    </repositories>

    <dependencies>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <version>1.18.2</version>
        </dependency>
        <dependency>
            <groupId>com.tngtech.jgiven</groupId>
            <artifactId>jgiven-junit</artifactId>
            <version>0.17.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-core</artifactId>
            <version>2.28.2</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.assertj</groupId>
            <artifactId>assertj-core</artifactId>
            <version>3.11.1</version>
```

```
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>pl.pragmatists</groupId>
            <artifactId>JUnitParams</artifactId>
            <version>1.1.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>1.7.21</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <pluginManagement>
            <plugins>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-compiler-plugin</artifactId>
                    <version>3.8.1</version>
                    <configuration>
                        <source>8</source>
                        <target>8</target>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-surefire-plugin</artifactId>
                    <version>3.0.0-M3</version>
                    <configuration>
                        <systemPropertyVariables>
                            <jgiven.report.text.color>true</jgiven.report.text.color>
                        </systemPropertyVariables>
                    </configuration>
                </plugin>
            </plugins>
        </pluginManagement>
    </build>
</project>
```