Programowanie równoległe i rozproszone
Kierunek Informatyka, WE, sem. VII

Laboratorium nr 5

Celem zajęć projektowanie współbieżnych i rozproszonych aplikacji JAVA opartych na środowisku Akka.

Aby zaliczyć laboratorium **przygotuj sprawozdanie**, w którym umieść odpowiedzi do poleceń.
Sprawozdanie **prześlij do odpowiedniego ćwiczenia** w iliasie.

Otwórz nowy projekt
File->New->Java Project
podaj nazwę projektu, zaakceptuj (na kolejnych ekranach po prostu akceptuj propozycje, również z tworzeniem module-info.java).

Zmień projekt na Maven (z menu pod prawym klawiszem myszy na nazwie projektu): Project->Configure->Convert to Maven project, zaakceptuj bez zmian.

W pliku pom.xml dodaj przed linjką </project>:

```
<dependencies>
	<dependency>
		<groupId>com.typesafe.akka</groupId>
		<artifactId>akka-actor_2.11</artifactId>
		<version>2.4-M2</version>
	</dependency>
</dependencies>
```

Zapisz plik pom.xml, zamknij.

Uwaga: prawdopodobnie w czasie zajęć są już dostępne nowsze wersje, możesz to sprawdzić w https://mvnrepository.com/

W projekcie dodaj pakiet "Przyklad1".

Będziemy tworzyć taką samą aplikację jak poprzednio: bank, w którym klienci korzystają ze wspólnego konta, ale tym razem skorzystamy z akka do obsługi komunikacji między klientami i tymże kontem w banku.

Poczytaj w sieci co to jest akka!
Odpowiedź umieść w sprawozdaniu.

Uwaga: niedawno do akka dodano lepszą obsługę (sprawdzanie) typów. Nasz pierwszy przykład będzie w klasycznym modelu, bez ścisłego typowania. Oznacza to między innymi, że wszystkie używane importowane klasy będą bez "typed", na przykład:
import akka.actor.ActorRef;
a nie
import akka.actor.typed.ActorRef;

Obiekty w aplikacji będą maksymalnie uproszczone:
- Bank nadal jest klasą statyczną, i posłuży nam tylko do stworzenia kont.
- Klienci będą się komunikować bezpośrednio z kontem - bez pośrednictwa banku. W rozbudowanej aplikacji obiekty powinny być zdefiniowane w sposób bardziej odpowiadający rzeczywistości - bank otrzymuje polecenie wpłaty/wypłaty, identyfikuje obiekt konta, sprawdza różne warunki (tożsamość, stan konta), i dopiero przekazuje polecenie zmiany stanu konta.
- Zaczynamy, jak poprzednio, od działania w nieskończonej pętli.

Projekt „SezAM wiedzy, kompetencji i umiejętności" jest współfinansowany przez Unię Europejską ze środków Europejskiego Funduszu Społecznego w ramach Programu Operacyjnego Wiedza Edukacja Rozwój

W modelu korzystającym z akka klasy pola/metody publiczne nie mają sensu (zastanów się dlaczego). Przekazywanie danych odbywa się za pomocą przesyłania wiadomości.

Docelowo będziemy chcieli obsłużyć następujące przepływy informacji:

Osoba -> KontoBankowe:
napis zawierający słowo "stan" === prośba o podanie stanu konta
liczba rzeczywista oznaczająca wysokość wpłaty (jeśli dodatnia) lub wypłaty (jeśli ujemna)

KontoBankowe -> Osoba:
liczba rzeczywista oznaczająca stan konta

Polecenia:

1. Utwórz aplikację bankową o następujących klasach:

Klasa Osoba
---
rozszerza UntypedActor
ma dwa pola prywatne:
String nazwisko i ActorRef konto

konstruktor z parametrami: nazwisko i konto

metoda operacje - prywatna, bez wyniku, bez parametrów
- losujemy kwotę
- wpłacamy tę kwotę na konto - konto.tell(kwota, self());
- wypłacamy tę samą kwotę (ta sama komenda, ale dla ujemnej wartości)
- piszemy komunikat o wykonanych operacjach (odpowiedni println)
- wysyłamy do konta prośbę o "stan konta" (tell w którym pierwszym parametrem jest odpowiedni tekst)

metoda onReceive (zobacz przykłady w sieci) - musi obsłużyć informację o nadesłanym stanie konta, po otrzymaniu stanu konta proszę po prostu wydrukować tę kwotę.

Klasa KontoBankowe:
---
rozszerza UntypedActor
ma trzy pola prywatne:
int numerKonta, double kwota, String nazwiskoWlasciciela

konstruktor z parametrami numer i nazwisko,
w konstruktorze można dodać odpowiedni wydruk, że zostało utworzone konto dla danej osoby

prywatne metody wpłata i wypłata podanej w parametrze kwoty (tu sprawdzenie, czy nie wypłacamy zbyt dużej kwoty)

metoda onReceive: zakładamy, że parametr msg może przyjąć wartości:
Integer: wówczas dla wartości większych od zera wołamy metodę wplata, dla mniejszych od zera - wyplata,
String: jeśli w przesłanym tekście występuje słowo "stan" odpowiadamy nadawcy wiadomością z podaniem kwoty na koncie:
getSender().tell(kwota, getSelf());

Klasa Bank
---
prywatne statyczna lista kont (ArrayList<ActorRef>)

publiczna statyczna funkcja noweKonto z parametrami ActorSystem system i String wlasciciel
w metodzie:
- jako nowy numer konta możemy przyjąc rozmiar listy kont
- tworzymy nowego aktora o typie KontoBankowe, nowym numerze konta i podanym wlascicielu dodajemy aktora do listy kont
- wynikiem działania funkcji jest nowo utworzony aktor

Program główny (klasa Main)
---

metoda main o typowym nagłówku

w main:
stworzenie systemu aktorów typu ActorSystem o dowolnej nazwie (np. bank)
stworzenie nowego konta z użyciem metody Bank.noweKonto z odpowiednimi parametrami

stworzenie trzech osób (aktorów) o nazwiskach "osoba 1", "osoba 2" i "osoba 3"

Sprawdź działanie programu.

Zauważ, że:
- program działa w nieskończoność (bo nigdzie nie ma usuwania aktorów), przy kolejnych uruchomieniach upewnij się, że zamknąłeś/zamknęłaś poprzednie,
- wpłaty i wypłaty nie mają miejsca (albo mają miejsce tylko raz - jeśli uruchomiłeś/aś operacje() w konstruktorze osoby.

Czy można dodać nieskończoną pętlę do Osoby, tak by wpłaty i wypłaty miały miejsce co sekundę? Spróbuj. Zdiagnozuj pojawiający się problem.

Zrzuty ekranu programu oraz swoje odpowiedzi zawrzyj w sprawozdaniu.

2. Dodanie do modelu cyklicznych odwołań (wykorzystanie schedule).

W klasie osoba dodaj metodę prestart(), na przykład:

```
@Override
public void preStart() {
        getContext().system().scheduler().schedule(
                        FiniteDuration.create(1, TimeUnit.SECONDS),
                        FiniteDuration.create(1, TimeUnit.SECONDS),
                        getSelf(),
                        "budzik",
                        getContext().dispatcher(),
                        null);
```

Odpowiedz na pytanie, czy w preStart zamiast wołania schedulera możemy wstawić kod:

```
Timer timer = new Timer();
timer.schedule(new TimerTask() {
   @Override
   public void run() {
      operacje();
```

```
        }
    }, 0, 1000);
```

Odpowiedź, zawrzyj w sprawozdaniu.

3. W metodzie onReceive klasy Osoby dodaj obsługę wiadomości "budzik". Nadejście takiej wiadomości ma skutkować wywołaniem operacje().

   W sprawozdaniu umieść zrzut ekranu kodu programu oraz okna działającego programu.

4. Rozbuduj program o sensowne brakujące elementy: powiąż konto z osobami, tak by nie wszyscy mieli do niego dostęp, rozbuduj działanie banku itp.

   Elementy, które dodałeś opisz w sprawozdaniu. Umieść kod programu dla tych dodatkowych elementów oraz okno działającego programu po zmianach.