Alfabet Morse'a

Zostałeś wybrany do bardzo ważnej misji; Twoim zadaniem jest umożliwić tajnym agentom (walczącym z "systemem") komunikację nie pozwalającą przechwycić przesyłanych informacji. Aby wróg nie przejął cennych danych agenci porozumiewają się za pomocą alfabetu Morse'a, ale jest on zaimplementowany w dziwny sposób. Każdy Agent implementuje następujący interfejs:

```
public interface TajnyAgent {
    void transmituj();
}
```

a każde wywołanie metody `transmituj()` powoduje przesłanie kolejnej kropki/kreski zakodowanej wiadomości. Jeśli `transmituj()` zakończy się pomyślnie jest to równoważne kresce a jeśli zakończy się wyjątkiem to kropce, chyba że jest to wyjątek:

- `RozmowaKontrolowana` który oznacza że ktoś podsłuchuje i trzeba zwrócić się o informację jeszcze raz;
- `Stop` który oznacza koniec litery;
- `Koniec` który oznacza koniec transferu informacji;
- `Zuber` który będzie wyjaśniony później.

Twoim zadaniem jest zaimplementowanie klasy `Agent` (implmentującej interfejsy `Readable` i `TajnyAgent`). Klasa `Agent` posiada cztery konstruktory:

- `Agent()`: agent nie nasłuchuje a nadaje w kółko tajne hasła: *pawiany/wchodzą/na/ściany!żyrafy/wchodzą/do/szafy!*;
- `Agent(String)`: agent nie nasłuchuje a nadaje raz `String` który dostał podczas konstrukcji;
- `Agent(TajnyAgent)`: agent nadaje tak samo jak w przypadku `Agent()` ale dodatkowo nasłuchuje co nadaje `TajnyAgent` i udostępnia rozkodowany tekst poprzez interfejs `Readable` (proszę kończyć `read()` po każdym znaku interpunkcyjnym)
- `Agent(TajnyAgent, String)`: który łączy funkcje dwóch poprzednich konstruktorów.

W poniższych dwóch testach :

```
import org.junit.Test;
import java.util.Scanner;
import static org.junit.Assert.assertEquals;

public class Test1 {
    @Test
    public void test1() {
        Scanner s = new Scanner(new Agent(new Agent("tajny/komunikat/państwa/podziemnego.")));
        assertEquals("tajny/komunikat/państwa/podziemnego.", s.next());
    }
}
```

```
import org.junit.Test;
import java.util.Scanner;
import static org.junit.Assert.assertEquals;

public class Test2 {
    @Test
    public void test2() {
        String[] tekst = String.valueOf("pawiany wchodzą na ściany żyrafy wchodzą do szafy " +
                "pawiany wchodzą na ściany żyrafy wchodzą do szafy " +
                "pawiany wchodzą").split("\\s+");
        Scanner s = new Scanner(new Agent(new Agent()));
        s.useDelimiter("([/!])");
        for (String str : tekst)
            assertEquals(str, s.next());
    }
}
```

jeden agent nadaje morsem tajny komunikat, a drugi go odczytuje, tłumaczy i udostępnia.

Najgroźniejszym wrogiem agentów jest pułkownik Zuber. Jeśli `TajnyAgent` nadający informacje zwróci wyjątek `Zuber` to:

- wyjątek ten nie liczy się jako kropka;
- `Agent` który go odczytał powinien nadać do końca aktualnie nadawane słowo, potem nadać słowo *zuber/*, a potem kontynuować nadawanie swojego normalnego komunikatu;
- kolejny wyjątek tego typu nie pojawi się zanim całe słowo *zuber/* nie zostanie odczytane.

```
import org.junit.Test;
import java.util.Scanner;
import static org.junit.Assert.*;

public class Test3 {
    @Test
    public void test3() {
        Agent agent = new Agent(new TajnyAgent() {
            int i;
            @Override
            public void transmituj() {
                if(i++ == 0) throw new Zuber();
                throw new Koniec();
            }
        });
        Scanner s = new Scanner(agent), t = new Scanner(new Agent(agent));
        t.useDelimiter("([/!])");
        assertEquals("pawiany",t.next());
        assertFalse(s.hasNext());
        assertEquals("zuber", t.next());
        assertEquals("wchodzą", t.next());
```

```
    }
}
```

Wszystkie definicje wyjątków, oraz definicja interfejsu `TajnyAgent` będą dostępne na satori. Należy submitować wyłącznie plik zawierający klasę `Agent` w pakiecie domyślnym.