WPROWADZENIE

Interface Definition Language (IDL)

Ważniejsze typy danych:

Typy podstawowe:

```
octet       short       long  long long   float
double      long double       char  string      boolean
```

Typy ograniczone, np.:

```
string<10> imie;
```

Wlasne typy, np.:

```
typedef short kolor;
typedef string imie;
```

Typy strukturalne, np.:

```
struct Osoba {
  string imie;
  string nazwisko;
  long   rok_urodzenia;
}
```

Tablice, np.:

```
typedef char narodowosc[2];
typedef Kolor tablica_kolorow[10][10];
```

Sekwencje:

```
typedef sequence<string> lista_nazwisk;
typedef sequence<string,6> kod_pocztowy;
typedef sequence<Osoba> lista_osob;
```

Interfejsy i metody

```
interface Osoba {
  attribute string imie;
  attribute string nazwisko;
};
interface Pracownik : Osoba {
  attribute long pensja;

  void Zwolnij();
  void ZwiekszPensje(in long podstawowa, in long premia);
  long SumaChorobowego(in long rok);
};
```

Moduły

```
module mojprogram {
  ...
  interface mojinterfejs {
    ...
  };
  ...
};
module pl {
  module edu {
    module pjwstk {
      interface mojprogram {
        ...
      };
    };
  };
};
```

Wyjątki

```
interface rachmistrz {
  exception dzielenie_przez_zero { };

  double podziel(in long dzielna, in long dzielnik)
              raises(dzielenie_przez_zero);
};
```

Java JDK

Do wykonania zadania potrzebne jest oprogramowanie Java SDK w wersji 8 lub starszej. Można pobrać je stąd: https://www.oracle.com/pl/java/technologies/javase/javase-jdk8-downloads.html

Program idlj

```
ildj [parametry] plik.idl
```

ważniejsze parametry:

- -f{client|server|all} - generuje kod klienta, servera, obu
- -OldImplBase - generuje kod dla BOA
- -td katalog - zapisuje generowany kod w podanym katalogu

Klasy generowane przez idlj

Dla przykładowego interfejsu:

```
interface Osoba {
  attribute string imie;
};
```

wywołanie `idlj -fall -OldImplBase osoba.idl` wygeneruje:

- _OsobaImplBase.java - klasa implementująca szkielet
- _OsobaStub.java - klasa implementująca pieniek
- OsobaOperations.java - interfejs zawierający deklaracje atrybutów i metod jako kod Javy
- Osoba.java - interfejs dziedziczący z OsobaOperations (tym interfejsem posługujemy się w programach Java)
- OsobaHelper.java - pomocnicze metod, głównie narrow
- OsobaHolder.java - "proteza" na argumenty out i inout z IDL, które nie dają się łatwo zmapować na Jave

ZADANIE WPROWADZAJĄCE

1. Utwórz plik Arytmetyka.idl I umieść w nim kod

```
interface Arytmetyka {
      attribute double s1, s2, wynik;

      exception DzieleniePrzezZero { };

      void suma();
      void roznica();
      void iloczyn();
      void iloraz() raises(DzieleniePrzezZero);
};
```

2. Wywołaj polecenie:

```
idlj -fall -OldImplBase Arytmetyka.idl
```

3. Umieść wygenerowane pliki w nowym projekcie Eclipse
4. Stworzyć klasę ArytmetykaServant dziedziczącą z _arytmetykaImplBase. Zaimplementować w niej wszystkie podziedziczone metody. Zmienne z zadeklarowanych atrybutów (s1, s2, wynik) zadeklarować lokalnie. Pamiętaj o obsłudze

niedozwolonych działań (np. dzielenia przez zero), w takim wypadku należy wyrzucić wyjątek.

5. Utworzyć klasę Server z metodą main. Zainicjować w niej obsługę ORB oraz utworzyć obiekt serwanta i podłączyć do ORB:

```
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args,null);

ArytmetykaServant as = new ArytmetykaServant();
orb.connect(as);
```

6. Zapisać do pliku referencję do obiektu na ORB

```
PrintWriter out = new PrintWriter(new BufferedWriter(
                          new FileWriter("ref.ior")));
out.println(
            orb.object_to_string(as) );
out.close();
```

7. Dodać kod czekający na wywołania ze strony klientów

```
java.lang.Object sync = new java.lang.Object();
synchronized (sync) {
              sync.wait();
}
```

8. Utworzyć klasę Client z metodą main. Zainicjować w niej obsługę ORB tak samo jak w serwerze

```
org.omg.CORBA.ORB orb = org.omg.CORBA.ORB.init(args, null);
```

9. Odczytać z pliku referencję zapisaną przez serwer

```
FileReader fr = new FileReader("ref.ior");
BufferedReader br = new BufferedReader(fr);
String ior = br.readLine();
```

10. Wykorzystać referencję do otrzymania zdalnego obiektu

```
org.omg.CORBA.Object obj = orb.string_to_object(ior);

arytmetyka proxy = arytmetykaHelper.narrow(obj);
```

11. Od teraz można korzystać z obiektu proxy tak jakby obiekt arytmetyka był obiektem lokalnym. Możemy wywołać np.:

```
proxy.s1(1);
proxy.s2(2);
proxy.suma();
System.out.println(proxy.wynik());
```

12. Uruchomić serwer, następnie uruchomić klienta i sprawdzić czy komunikacja działa.

13. Zastąp wymianę referencji poprzez pliki wymianą z użyciem usługi nazwowej.

USŁUGA NAZWOWA

Usługa nazwowa pozwala wyeliminować potrzebę przenoszenia IOR z klienta na serwer jako ciąg znaków. Zamiast tego, obiekt jest rejestrowany w osobnym serwerze, pod dowolnie wybraną, stałą nazwą.

Aby uruchomić usługę nazwową należy wywołać z konsoli program `tnameserv`.

Wszystkie potrzebne klasy znajdziemy w pakiecie `org.omg.CosNaming.*;`
Aby uzyskać dostęp do usługi nazwowej należy:

- Uzyskać referencję do obiektu usługi nazwowej:

```
org.omg.CORBA.Object objRef = orb.resolve_initial_references("NameService");
```

- Zrzutować obiekt na interfejs `NamingContext`:

```
NamingContext ncRef = NamingContextHelper.narrow(objRef);
```

- Stworzyć komponent nazwowy:

```
NameComponent nc = new NameComponent("Arytmetyka", "");
```

- Utworzyć ścieżkę nazwową (tablica komponentów nazwowych):

```
NameComponent path[] = {nc};
```

Następnie, dla części **serwerowej** należy zarejestrować serwanta poprzez związanie go z wybraną nazwą:

```
ncRef.rebind(path, as);
```

Dla **klienta**, należy pobrać referencję na podstawie ścieżki nazwowej i zrzutować do interfejsu.

```
arytmetyka proxy = arytmetykaHelper.narrow(ncRef.resolve(path));
```

ZADANIE NA OCENĘ

Korzystając z usługi nazwowej i złożonych typów danych (struktur) zbuduj z użyciem standardu CORBA własną usługę podobną do tej, stworzonej w ramach zadania SOAP. Usługa powinna zawierać co najmniej dwie metody biznesowe i korzystać z własnego, złożonego modelu danych.