

KONFIGURACJA ŚRODOWISKA

1. Utwórz nowy projekt javowy w środowisku IntelliJ IDEA.
2. Sprawdź, czy mapper języka IDL na Javę jest prawidłowo skonfigurowany. Wpisz w terminalu:

idlj

3. Jeżeli wyświetli się komunikat:

idlj is not recognized as an internal or external command, operable program or batch file,

to znak, że ścieżka do mappera nie jest poprawnie zdefiniowana. Aby program właściwie działał należy w terminalu wpisać:

set "path=%path%;ścieżka_do_folderu_w_którym_znajduje_się_idlj"

4. Sprawdź ponownie stan mappera idlj. W razie potrzeby zrestartuj środowisko.

IMPLEMENTACJA SERWERA W CORBA

1. Utwórz nowy plik w katalogu *src* i nazwij go *Hello.idl*
2. Wypełnij go następującą zawartością:

```
module HelloApp {  
    interface Hello {  
        string sayHello();  
    };  
};
```

3. Powyższy kod napisany jest w standardowym języku IDL, z którego za moment będziemy chcieli wygenerować pliki w języku Java.
4. Aby to zrobić upewnij się, że znajdujesz się w katalogu zawierającym plik *Hello.idl*, a następnie wpisz w terminalu komendę:

idlj -fall Hello.idl

5. Powinien utworzyć się katalog *HelloApp* zawierający sześć plików:
 - *Hello*(wygenerowana wersja javy z IDL)
 - *HelloHelper*(odpowiedzialna za czytanie i pisanie danych do strumieni CORBA)
 - *HelloHolder*(przechowuje publiczną instancję klasy *Hello*)
 - *HelloOperations*(zawiera zadeklarowane wcześniej metody interfejsu *Hello*)
 - *_HelloStub*(client stub)
 - *HelloPOA*(server skeleton)

6. Chcielibyśmy dostosować zwracane wartości metod wygenerowanych z kontraktu IDL wg własnych potrzeb. Aby to zrobić stworzymy nową klasę i nazwiemy ją *HelloImpl.java* w katalogu *HelloApp*. Będzie to nasza autorska implementacja spełniająca warunki wcześniej utworzonego kontraktu. Nowo utworzony plik wypełniamy kodem:

```
public class HelloImpl extends HelloPOA {
    @Override
    public String sayHello() {
        return "Welcome";
    }
}
```

7. Nadszedł czas na utworzenie klasy zawierającej kod naszego serwera. W tym celu tworzymy nowy plik *HelloServer.java* w katalogu *HelloApp* oraz umieszczamy w nim co następuje:

```
public class HelloServer {
    public static void main(String args[]) {
        Properties orbprop = new Properties();
        orbprop.put("org.omg.CORBA.ORBInitialPort", "2004");
        ORB orb = ORB.init(args, orbprop); //utworzenie i inicjalizacja ORB
        try {
            POA rootpoa =
                POAHelper.narrow(orb.resolve_initial_references("RootPOA"));
            rootpoa.the_POAManager().activate();
            HelloImpl helloImpl = new HelloImpl();
            org.omg.CORBA.Object ref = rootpoa.servant_to_reference(helloImpl);
            Hello href = HelloHelper.narrow(ref);
            org.omg.CORBA.Object objRef =
                orb.resolve_initial_references("NameService"); //pobranie kontekstu nazw
            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
            String name = "Hello";
            NameComponent path[] = ncRef.to_name(name); //wyszukanie obiektu w
            kontekście
            ncRef.rebind(path, href);
            System.out.println("Start serwera...");
            orb.run();
        } catch (InvalidName | AdapterInactive | ServantNotActive | WrongPolicy
            | CannotProceed | org.omg.CosNaming.NamingContextPackage.InvalidName | NotFound
            | invalidName) {
            invalidName.printStackTrace();
        }
    }
}
```


8. Aby wystartować serwer musimy najpierw skompilować wygenerowane klasy z folderu *HelloApp*. W tym celu upewniamy się, że znajdujemy się w folderze zawierającym plik *HelloServer.java* i wpisujemy w terminalu:

```
javac HelloServer.java *.java
```

W katalogu bieżącym(po jego odświeżeniu) powinny pojawić się nowe pliki z rozszerzeniem *.class*. Jeżeli tak się nie stało powtórz krok 8.

9. Przed wystartowaniem serwera należy uruchomić usługę orbd. Przechodzimy do folderu wyżej(polecenie *cd..*), a następnie wpisujemy w terminalu:

orbd -ORBInitialPort 2004

10. Prawidłowe wykonanie poprzedniego kroku skutkuje wygenerowaniem folderu orb.db oraz zablokowanie możliwości pisania w obecnej konsoli. Jeżeli tak się nie stało, powtórz krok 9.
11. Ostatnim krokiem jest wystartowanie serwera. W tym celu uruchamiamy nowe okno terminala(ikona ), upewniamy się, że jesteśmy w folderze *src* i wpisujemy w konsoli:

java HelloApp/HelloServer

12. W konsoli powinien wyświetlić się napis: *Start serwera...*

IMPLEMENTACJA KLIENTA W CORBA

1. W celu napisania klienta utworzymy nowy projekt javowy.
2. Operacje, które klient będzie wykonywać na serwerze muszą być zgodne z wcześniej stworzonym kontraktem IDL, konieczny więc będzie dostęp do niego. Kopiujemy plik *Hello.idl* z poprzedniego projektu i umieszczamy w katalogu *src* bieżącego projektu.
3. Generujemy klasy javowe z pliku IDL:

idlj -fall Hello.idl

4. Pora na główny kod klienta. Tworzymy plik *HelloClient.java* w nowo utworzonym katalogu *HelloApp*, a następnie wypełniamy go kodem:

```
public class HelloClient{
    public static void main(String args[]){
        try{
            Properties orbProp = new Properties();
            orbProp.put("org.omg.CORBA.ORBInitialPort", "2004");
            ORB orb = ORB.init(args, orbProp); // utworzenie i inicjalizacja ORB
            org.omg.CORBA.Object objRef = orb.resolve_initial_references
                ("NameService"); // pobranie kontekstu nazw
            NamingContextExt ncRef = NamingContextExtHelper.narrow(objRef);
            String name = "Hello"; // wyszukanie obiektu w kontekście
            Hello helloImpl = HelloHelper.narrow(ncRef.resolve_str(name));
            System.out.println(helloImpl.sayHello());
        } catch (Exception e) {
            System.out.println("ERROR : " + e) ;
        }
    }
}
```

5. Podobnie jak podczas tworzenia serwera musimy skompilować nowo zaimplementowane klasy. W tym celu będąc w katalogu *HelloApp* wpisujemy w konsoli:

*javac HelloClient.java *.java*

6. Aby uruchomić naszego klienta przechodzimy folder wyżej i wpisujemy w konsoli:

java HelloApp/HelloClient

7. Na konsoli powinien pojawić się napis: *Welcome*

Zadania do samodzielnego rozwiązania

1. Zmodyfikuj serwer i klienta w taki sposób, aby metoda *sayHello()* przyjmowała za argument Twoje imię i zwracała powitanie Twojej osoby.

WSKAZÓWKA #1

Plik *Hello.idl* to kontrakt, do którego dostęp musi mieć zarówno klient, jak i serwer.

WSKAZÓWKA #2

Język IDL zawiera 3 parametry, która mogą być umieszczone przy argumentach metod określając kierunek przesyłania parametru:

in – client -> serwer
out – serwer -> client
inout – client <-> serwer

np. string hello(in string name);

WSKAZÓWKA #3

Pamiętaj o kompilacji po zmianach w kodzie plików i usunięciu starych plików *.class*.

2. Zmodyfikuj serwer i klienta w taki sposób, aby serwer zwracał cały obiekt(wcześniej zdefiniowany w kontrakcie i wypełniony przykładowymi danymi), klient odbierał ten obiekt i wyświetlał na konsoli jego atrybuty, a następnie zmieniał je(klient) na serwerze i ponownie wyświetlał.

WSKAZÓWKA #1

W języku IDL nowe struktury deklaruje się podobnie jak w C/C++, np.:

```
struct PersonDetails {  
    string    name;  
    string    surname;  
    long      age;  
};
```

WSKAZÓWKA #2

Jeżeli nową strukturę zadeklaruje się przy pomocy słowa kluczowego *attribute*, to idlj wygeneruje getter i setter dla tej struktury.

WSKAZÓWKA #3

Wszystkie pola w IDL w danej strukturze są zawsze publiczne.