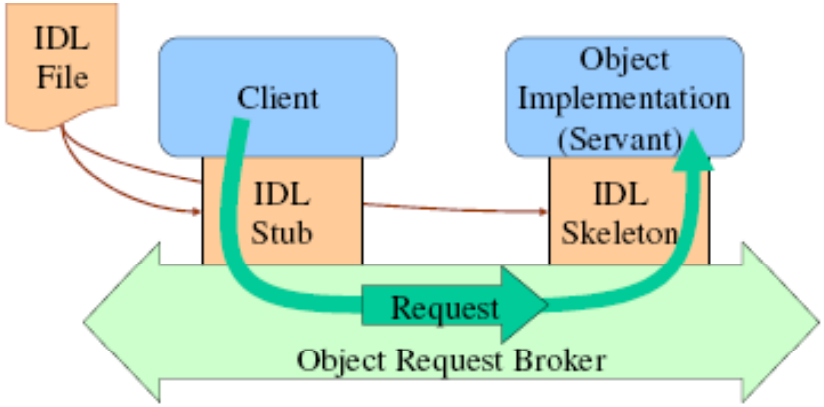
Lecture (10)
CORBA

Dr. Ahmed ElShafee

Agenda

- Xx
- Xx
- xx

Application Diagram



Development Steps

1) Design the Application
2) IDL Specification
3) IDL Compilation (Code Generation)
4) Write the Client & Server implementation specific code
5) Compile the source code
6) Run the application

IDL

Parameters

1) Java uses pass-by-value for parameters (including parameters that are references)
2) IDL has **in**, **out** and **inout** types of parameters
3) The **in** parameter type maps to a normal Java parameter since it does not need to be changed
4) **out** and **inout** parameter types are passed via instances of Java Holder classes

Holder Classes

Holder classes encapsulate the real value of a parameter in case if using **out** or **inout** parameters

Why? To overcome java's pass by value,

```
// user code

// select a target object
Example.Modes target = …;
// prepare to receive out
IntHolder outHolder = new IntHolder();
// set up the in side of the inout
IntHolder inoutHolder = new IntHolder (131);
// make the invocation
int result = target.operation (
                    outHolder, inoutHolder);
// use the values of holders
…outHolder.value…
…inoutHoulder.value…
```

```
// generated java
package Example;
public interface Modes {
  int operation (IntHolder outArg,
                 IntHolder inoutArg);
}
```


Helper Classes

1) all user-defined IDL types have a Helper Java class
2) insert and extract Any
3) get CORBA::TypeCode of the type
4) narrow (for interfaces only)

// Helper classes are a term given to classes that are used to assist in providing some functionality, though that functionality isn't the main goal of the application.

Data type

IDL Type	Java Type
boolean	boolean
char	char
octet	byte
string	java.lang.String
short	short
unsigned short	short
long	int
unsigned long	int
long long	long
unsigned long long	long
float	float
double	double

IDL to Java

- IDL char
```
const char MyChar = 'A';
```
- Java char
```
final public class MyChar
{
final public static char
value = (char)'A';
}
```

- IDL octet
```
void foo(in octet x);
```
- Java byte
```
public void foo(byte x);
```

the **final** keyword is used to define an entity which cannot later be changed. A **final method** cannot be overridden by subclasses.

- IDL boolean
```
const boolean truth =
TRUE;
```
- Java boolean
```
final public class truth
{
    final public static
boolean value = true;
}
```
- IDL constants `TRUE` and `FALSE`
- Java constants `true` and `false`

- IDL integers
  - (unsigned) short
  - (unsigned) long
  - (unsigned) long long?

```
const unsigned short
MyUnsignedShort = 1580;
```

- Java integers
  - short
  - int
  - long

```
final public class
MyUnsignedShort
{
       final public
static short value =
(short)1580;
}
```


- IDL floating-point float, double

```
const double MyDouble =
1.23456789;
```

- Java floating-point *float, double*

```
final public class
MyDouble
{
          final public
static
    double value =

    (double)1.23456789;
}
```

- IDL Enum

```
enum MyEnum
(none,first,second);
```

- Java class

```
final public class MyEnum
{
 final public static int
none = 0;
 final public static int
first = 1;
 final public static int
second = 2;
 final public static int
narrow(int i) throws
CORBA.BAD_PARAM (..);
}
```

- the narrow method is for checking enum values

- IDL *struct*

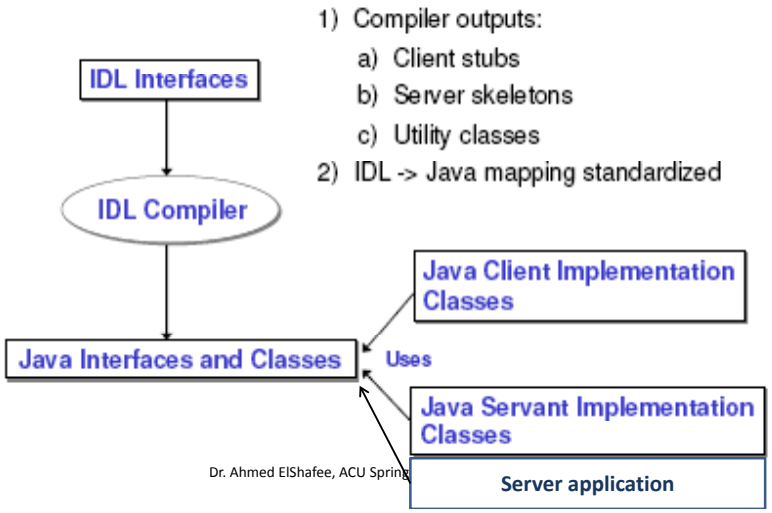
```
struct MyStruct
(
 long mylong;
 string mystring;
);
```

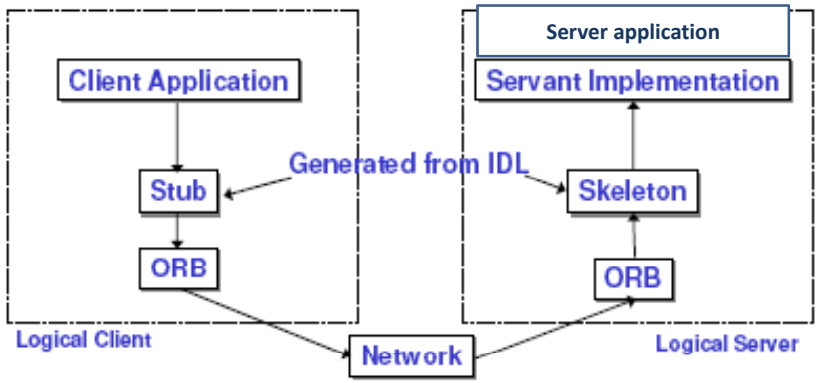
- Java *class*

```
final public class
MyStruct
(
    public MyStruct(int
_mylong, String
_mystring) {…};
    public MyStruct()
{…};

    public int mylong;
    public String
mystring;
}
```


The Big Picture





Example 01

This presents a file download CORBA application

The server provide a calculator service, and client request that service through its published name

There are a number of steps involved:

1) Define an interface in IDL
2) Map the IDL interface to Java (done automatically)
3) Implement the interface
4) Develop the server
5) Develop a client
6) Run the naming service, the server, and the client.

Example 01>> interface

```
interface CalculatorInterface
    {
    long long add(in long long a, in long long b);
    long long sub(in long long a, in long long b);
    long long mul(in long long a, in long long b);
    long long div(in long long a, in long long b);
    };
```


Example 01>> interface compiling

> idlj -fall -oldImplBase CalculatorInterface.idl

_CalculatorInterfaceImplBase.java
_CalculatorInterfaceStub.java
CalculatorInterface.java
CalculatorInterfaceHelper.java
CalculatorInterfaceHolder.java
CalculatorInterfaceOperations.java

> javac CalculatorInterface.java

> javac CalculatorInterfaceHolder.java

> javac _CalculatorInterfaceImplBase.java

Example 01>> servant

```
import java.io.*;
public class CalculatorServant extends _CalculatorInterfaceImplBase
    {
    public long add(long a, long b)
            { return a + b; }
    public long sub(long a, long b)
            { return a - b; }
    public long mul(long a, long b)
            { return a * b; }
    public long div(long a, long b)
            { return a / b; }
    }
```

Example 01>> servant compiling

> javac CalculatorServant.java

CalculatorServant.class

Example 01>> server

```
import java.io.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class CalculatorServer
    {
    public static void main(String args[])
        {
        try
            {
            // create and initialize the ORB
            ORB orb = ORB.init(args, null);
            // create the servant and register it with ORB
            CalculatorServant CS = new CalculatorServant();
```



```
orb.connect(CS);
            // get the root naming context
            org.omg.CORBA.Object objRef
    =orb.resolve_initial_references("NameService");
            NamingContext ncRef
    =NamingContextHelper.narrow(objRef);
            // Bind the object reference in naming
            NameComponent nc = new
        NameComponent("CalculatorService", " ");
            NameComponent path[] = {nc};
            ncRef.rebind(path, CS);
            System.out.println("Server started....");
            // Wait for invocations from clients
            java.lang.Object sync = new java.lang.Object();
```



```
synchronized(sync)
                {
                sync.wait();
                }
            }
        catch(Exception e)
            {
            System.err.println("ERROR: " + e.getMessage());
            e.printStackTrace(System.out);
            }
        }
    }
```

Example 01>> server compiling

> javac CalculatorServer.java

CalculatorServer.class

Example 01>> client

```
import java.io.*;
import java.util.*;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;

public class CalculatorClient
    {
    public static void main(String argv[])
        {
        try
            {
            // create and initialize the ORB
            ORB orb = ORB.init(argv, null);
            // get the root naming context
```



```
org.omg.CORBA.Object objRef
    =orb.resolve_initial_references("NameService");
            NamingContext ncRef =
    NamingContextHelper.narrow(objRef);
            NameComponent nc = new
    NameComponent("CalculatorService", " ");
            // Resolve the object reference in naming
            NameComponent path[] = {nc};
            CalculatorInterfaceOperations CS =
    CalculatorInterfaceHelper.narrow(ncRef.resolve(path));
            System.out.println( "4+3="+CS.add(4, 3) );
            System.out.println( "4-5="+CS.sub(4, 5) );
            System.out.println( "9*6="+CS.mul(9, 6) );
            System.out.println( "12/3="+CS.div(12, 3) );
```



```
            }
        catch(Exception e)
            {
            System.out.println("FileClient Error: " + e.getMessage());
            e.printStackTrace();
            }
        }
    }
```

Example 01>> server compiling

> javac CalculatorClient.java

CalculatorClient.class

Example 01>> files distribution

client	server
CalculatorInterface.class	CalculatorInterface.class
_CalculatorInterfaceStub.class	_CalculatorInterfaceImplBase.class
CalculatorInterfaceHelper.class	CalculatorInterfaceHelper.class
CalculatorInterfaceOperations.class	CalculatorInterfaceHolder.class
CalculatorClient.class	CalculatorInterfaceOperations.class
	CalculatorServant.class
	CalculatorServer.class

Example 01>> running

Client	Server
java CalculatorClient c:\hello.txt -ORBInitialHost 10.3.3.114 -ORBInitialPort 2500	tnameserv -ORBInitialPort 2500
	java CalculatorServer -ORBInitialPort 2500

Example 02

This presents a file download CORBA application

The client request for a file and the server in turn sends the file to the client which then saves it on the local machine.

There are a number of steps involved:

1) Define an interface in IDL
2) Map the IDL interface to Java (done automatically)
3) Implement the interface
4) Develop the server
5) Develop a client
6) Run the naming service, the server, and the client.

Example 02>> FileInterface.idl (Interface)

```
interface FileInterface
    }
    typedef sequence<octet> Data;
    Data downloadFile(in string fileName);
    {
```


map the IDL interface to Java

```
>idlj -fall -oldimplbase FileInterface.idl
```

Will generate

FileInterfacePackage/DataHelper.java
FileInterface.java
FileInterfaceHelper.java
FileInterfaceHolder.java
FileInterfaceOperations.java
_FileInterfaceImplBase.java
_FileInterfaceStub.java


```
> javac FileInterface.java
> javac FileInterfaceHolder.java
> javac _FileInterfaceImplBase.java
```

FileInterfacePackage/DataHelper.class
FileInterface.class
FileInterfaceOperations.class
_FileInterfaceImplBase.class

Example 02>> FileServant.java (implementation)

```
import java.io.*;
public class FileServant extends _FileInterfaceImplBase
    {
    public byte[] downloadFile(String fileName)
            {
            File file = new File(fileName);
            byte buffer[] = new byte[(int)file.length()];
            try
                    {
                    BufferedInputStream input = new
                    BufferedInputStream(new FileInputStream(fileName));
```

```
input.read(buffer,0,buffer.length);
                    input.close();
                    }
          catch(Exception e)
                    {
                    System.out.println("FileServant Error:"+e.getMessage());
                    e.printStackTrace();
                    }
          return(buffer);
    }
}
```


> javac FileServant.java

FileServant.class

Example 02>> FileServer.java (server program)

- Create ORB object called orb
- Create FileServent object called fileRef
- Connect fileRef 2 orb
- Create an CORBA object called ObjRef, looks like directory called "NameService"
- The directory contains list of services and their corresponding object that provide the services.
- Directory contents class called NamingContext, we create object called (ncRef)
- We will add array of records to ncRef, class is NameComponent, object called path (actualy array of objects) (contains an object called nc)
- Nc object contains 2 fields (service id, service kind)
- Bind ncRef (services directory) to fileRef (object itself)


```
import java.io.*;
import org.omg.CosNaming.*;
import org.omg.CosNaming.NamingContextPackage.*;
import org.omg.CORBA.*;
public class FileServer
    {
    public static void main(String args[])
          {
          try
                    {
```

```
        // create and initialize the ORB
        ORB orb = ORB.init(args, null);
        // create the servant and register it with ORB
        FileServant fileRef = new FileServant();
        orb.connect(fileRef);
        // get the root naming context
        org.omg.CORBA.Object objRef
=orb.resolve_initial_references("NameService");
        NamingContext ncRef
=NamingContextHelper.narrow(objRef);
```



```
        // Bind the object reference in naming
        NameComponent nc = new
NameComponent("FileTransfer", " "); //id , kind
        NameComponent path[] = {nc};
        ncRef.rebind(path, fileRef);
        System.out.println("Server started....");
        // Wait for invocations from clients
        java.lang.Object sync = new java.lang.Object();
        synchronized(sync) // to synchronize shared objects   {
                sync.wait(); //communication between callers
                        }}
```



```
catch(Exception e)
        {
        System.err.println("ERROR: " + e.getMessage());
        e.printStackTrace(System.out);
        }
    }
}
```



```
> javac FileServer.java
```

FileServer.class

Example 02>> FileClient.java (client program)

```
import java.io.*;
import java.util.*;
import org.omg.CosNaming.*;
import org.omg.CORBA.*;

public class FileClient
    {
    public static void main(String argv[])
        {
```



```
try
                {
                // create and initialize the ORB
                ORB orb = ORB.init(argv, null);
                // get the root naming context
                org.omg.CORBA.Object objRef
    =orb.resolve_initial_references("NameService");
                NamingContext ncRef =
    NamingContextHelper.narrow(objRef);
                NameComponent nc = new
    NameComponent("FileTransfer", " ");
```



```
// Resolve the object reference in naming
                NameComponent path[] = {nc};
                FileInterfaceOperations fileRef =
    FileInterfaceHelper.narrow(ncRef.resolve(path));
                if(argv.length < 1)
                        {
                        System.out.println("Usage: java FileClient
    filename");
                        }
```



```
// save the file
                File file = new File(argv[0]);
                byte data[] = fileRef.downloadFile(argv[0]);
                BufferedOutputStream output = new
    BufferedOutputStream(new FileOutputStream(argv[0]));
                output.write(data, 0, data.length);
                output.flush();
                output.close();
                }
```

```
catch(Exception e)
                {
                System.out.println("FileClient Error: " + e.getMessage());
                e.printStackTrace();
                }
        }
    }
```


> javac FileClient.java

FileClient.class

Files distribution

client	server
FileInterfacePackage/DataHelper.class	FileInterfacePackage/DataHelper.class
FileInterface.class	FileInterface.class
_FileInterfaceStub.class	_FileInterfaceImplBase.class
FileInterfaceHelper.class	FileInterfaceHelper.class
FileInterfaceOperations.class	FileInterfaceHolder.class
FileClient.class	FileInterfaceOperations.class
	FileServant.class
	FileServer.class

Example 02>> Running

Client	Server
java FileClient c:\hello.txt -ORBInitialHost 10.3.3.114 -ORBInitialPort 2500	tnameserv -ORBInitialPort 2500
	java FileServer -ORBInitialPort 2500

Thanks,

See you next Week, isA