Java programming environment for Distributed Programming

M. Sato

What's Java

- A Java program is defined as a set of class definitions. A function definition or a data definition outside class definition as in C is not allowed.
- Object-oriented programming language. Member functions (methods), private/public attribute for members, inheritance.
- Constructor, but no destructor is needed. When an object is not referenced any more, it will be reclaimed by the garbage collection.
- No pointer notation. That is, all object is represented through a pointer from C++ point of view (reference type). In C++, all member functions of Java are virtual member functions.
- No multiple inheritance. Only single inheritance is provided.
- Definition of interface is virtual abstract class in C++.
- No Operator overloading. Template are introduced in later versions of Java.

What is C++

- Class concept is introduced into C to define an object. An object is defined as a set of data with methods to access and manipulate these data. Struct in C is a defined as a object with only public data.
- Inheritance allows one data type to acquire properties of other data types. Inheritance can be defined as a relation between classes. Public methods/member is visible (accessable) from other class. Private methods/members are not. Multiple inheritance (inheritance from more than one classes) is supported
- Constructor to allocate class objects. Destructor to de-allocate objects. These methods are defined as member methods, and called when allocate/decallocate objects. new / delete   Operator
- Virtual method function
- Operator overloading, an infix operators can be defined for objects.
- Function overloading allows programs to declare multiple functions having the same name (but with different arguments). int foo(int x) and int foo（double） are different functions. Implicit type conversion
- Default argument. Argument passing by reference.
- C++ templates enable generic programming.

Comments on Java

- Java is sometime compared with C++, but the basic concept of Java is similar to that of Smalltalk.
- A Java program is usually compiled into Java byte-code which run on Java Virtual Machine (Java VM)
  - Java VM is a byte code interpreter.
  - Just-In-Time (JIT) compiler is used to speedup the execution. JIT compiles byte code to native code at runtime.
- This configuration and machanism of execution give the flexibility and usability as a language for “internet”.

Very simple example of Java

- example
  - test.java : Java source file
  - Compile by “javac test.java” into test.class
  - Execute program by “java test”. Start execution from satatic main method in test.class
  - The declaration “static” means the static method is shared by the different object of the same class.

```
class test {
 public static void main(String arg[]){
   System.out.println(“hello”);
 }
}
```

Simple example of java

- More example

```
class test {
 public static void main(String arg[]){
   hello h = new hello(“jack”);
   h.say();
 }
}
```

```
class hello {
 String who;
 public hello(String who){
   this.who = who;
}
 public void say(){
   System.out.println(“hello”+who);
}
}
```

Simple example of java

- More more example, example using inheritance

```
class test {
 public static void main(String arg[]){
   hello h = new konnichwa(“jack”);
   h.say();
 }
}
```

```
class konnichiwa extends hello {
 String who;
 public konnichiwa(String who){
   this.who = who;
}
 public void say(){
   System.out.println(“konnnichwa”+who);
}
}
```

Simple example of java

- Class path
  - The environment variable to specify paths to find class (file). It may be directory or Jar file
  - Jar file is a ziped file containing class files in directory form.
  - Java VM load class file at run-time.
- Interface definition
  - Specify the member function (method) which have to be defined in sub class.
  - Java support only a single inheritance, but class definition can have multiple interface.

Simple example of java

- More more example, example using interface

```
class test {
 public static void main(String arg[]){
   oval ov = new oval(…);
   draw_it(ov);
 }
 static void draw_it(drawable o){
    … o.draw(); …
 }
}
```

```
interface drawable {
  void draw();
}
…
class circle implements drawable{
  void draw();
}
class oval implements drawable{
  void draw();
}
class polygon implements drawable{
  void draw();
}
```

Principle of Object-oriented programming

- **Objective of object oriented design (different from programming language)**
  - **Maintenance-ability: making programs keeping easy to understand and modify during development and after. It is important from other to understand easily (readability).**
  - **Extend-ability: Adding other functions keep modification of other codes as small as possible.**
  - **Reuse-ability: using a part of codes as a component for other program. It increase the value of software.**
  - **Efficiency: must be faster.**

Some guidelines of Object-oriented design

- **Understand that the public inheritance is a relation of “ is a”.**
- **How and where to use “interface”. Understand the difference between interface and inheritance.**
- **Describe the “has-a” relation and the “is implemented of” relation by layering (item 40)**
- **Use private inheritance correctly (item 41)**

Distributed programming by Java

- RMI = Remote Method Invocation,

  a mechanism for programming distributed systems.
  - By using RMI, instances of objects can be allocated in different machines, and call methods of the objects allocated in different machine with each others, for constructing distributed systems.

Remote Procedure Call

- For programming distributed systems, TCP/IP and UDP are used as a low-level communication layer. But with such low-level communication layer, the programmer have to design the protocol of communication to implement each function/functionarity.

- RPC(remote procedure call) provides an abstraction of protocol as a function call.
  - SUN　RPC
  - CORBA (Java、C++)
  - Web Service
  - RMI, Jini….
  - JAX RPC
  - …

Basic of Network programming

- **Server side:**

```
s = socket(); /* create socket*/
bind(s,address); /* put name to it */
listen(s,backlog); /* Specifiy backlog */
ss = accept(s); /* when connection comming
          return a new file descriptor for it */
close(s); /* close s if it is needed any more */
recv(ss,…); /* start reading */
```

- **Client side:**

```
s = socket(); /* create socket */
connect(s,address); /* connection it*/
send(s,…); /* start sending */
```

Server side(1)

```
// 省略
int my_fd;
struct sockaddr_in my_sin;
static int _setup_server_socket(struct sockaddr_in *sinp,
                                int port, int backlog);
int main(int argc,char *argv[])
{
    int sinlen;
    struct sockaddr_in client_sin;
    char buf[128];
    int r,s;
    int port;
    if(argc != 2){
        fprintf(stderr,"%s #port¥n",argv[0]);
        exit(1);
    }
    port = atoi(argv[1]);
    printf("server test program ... wait on port %d¥n",port);
   my_fd = _setup_server_socket(&my_sin,port,1);
    sinlen = sizeof(struct sockaddr_in);
    s = accept(my_fd,(struct sockaddr *)&client_sin,&sinlen);
    if(s < 0){
        perror("accept failed");
        exit(1);
    }
    while((r = read(s,buf,128)) >= 0){
        write(1,buf,r);
    }
    printf("terminated ...¥n");
    close(s);
    close(my_fd);
    exit(0);
```

Server side(2)

```
static int _setup_server_socket(struct sockaddr_in *sinp,int port,
       int backlog)
{
    int sinlen,r;
    struct sockaddr_in sin;
    char hostname[MAXHOSTNAMELEN];
    struct hostent *hp;
    int fd;

    fd = socket(AF_INET, SOCK_STREAM, 0);
    if(fd < 0){
        perror("socket failed");
        exit(1);
    }
    r = gethostname(hostname,MAXHOSTNAMELEN);
    if(r < 0){
        perror("hostname");
        exit(1);
    }
    printf("hostname=%s¥n",hostname);

    hp = gethostbyname(hostname);
    if(hp == NULL){
        perror("gethostbyname");
        exit(1);
    }
    memset(&sin, 0, sizeof(sin));
    sin.sin_family = AF_INET;
    sin.sin_port = htons(port);
    bcopy(hp->h_addr,&sin.sin_addr.s_addr,hp->h_length);
```

Server side(3)

```
  sinlen = sizeof(sin);

    r = bind(fd, (struct sockaddr *) & sin, sizeof(sin));
    if (r < 0){
        perror("bind");
        exit(1);
    }

    r = listen(fd,backlog);  /* set backlog */
    if (r < 0){
        perror("listen");
        exit(1);
    }

     r = getsockname(fd,(struct sockaddr *)sinp, &sinlen);
    if(r < 0){
        perror("getsockname");
        exit(1);
    }

    return fd;
}
```

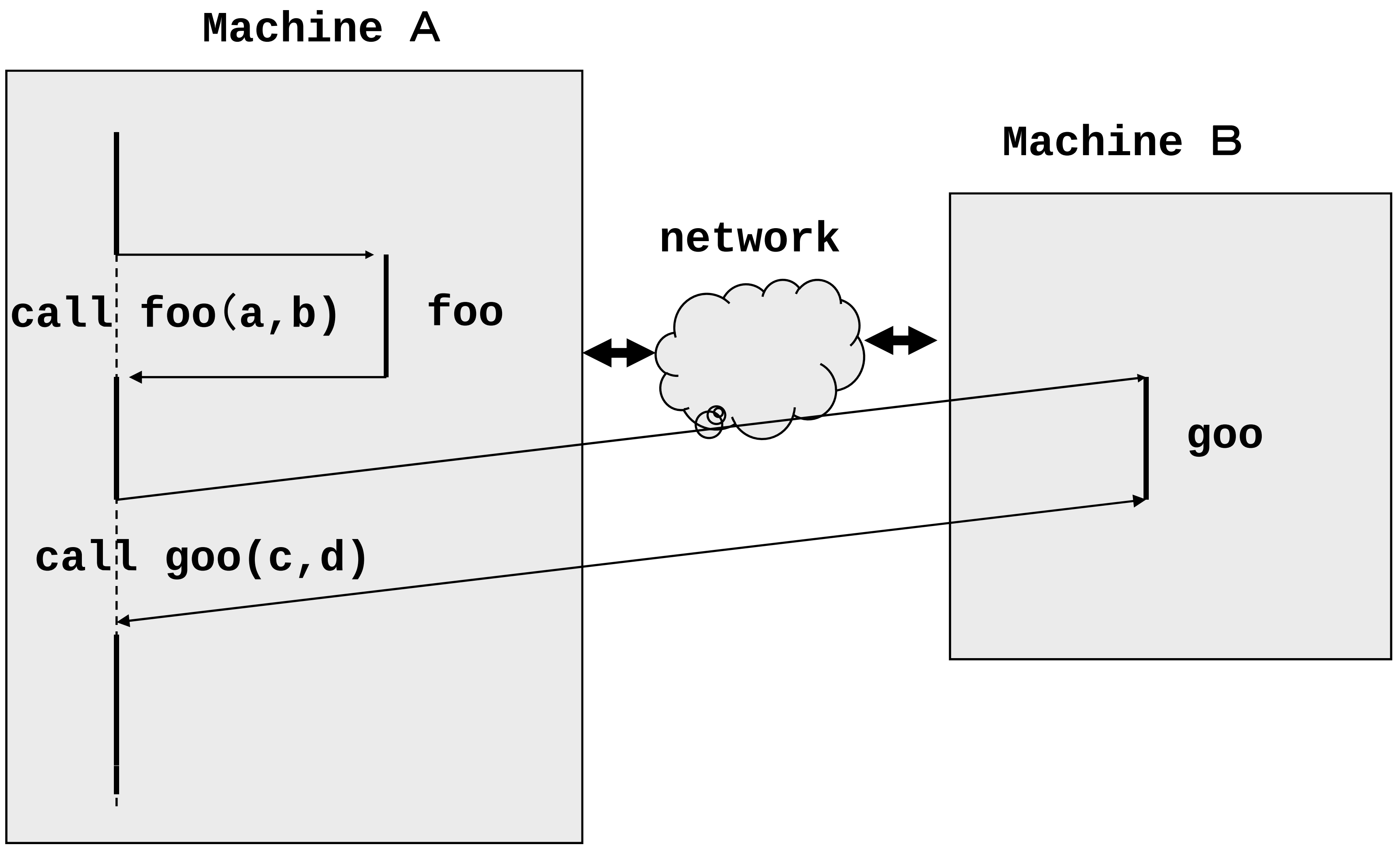
Client side(1)

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>

#ifndef MAXHOSTNAMELEN
#define MAXHOSTNAMELEN 256
#endif

int main(int argc,char *argv[])
{
    int r;
    struct sockaddr_in sin;
    char hostname[MAXHOSTNAMELEN];
    struct hostent *hp;
    int fd,port;
    char buf[128];
     if(argc != 3){
        fprintf(stderr,"%s: hostname port¥n");
        exit(1);
    }
    strcpy(hostname,argv[1]);
    port = atoi(argv[2]);
    printf("client test ... connect to %s:%d¥n",hostname,port);
    hp = gethostbyname(hostname);
    if(hp == NULL){
        perror("gethostbyname");
        exit(1);
    }
```

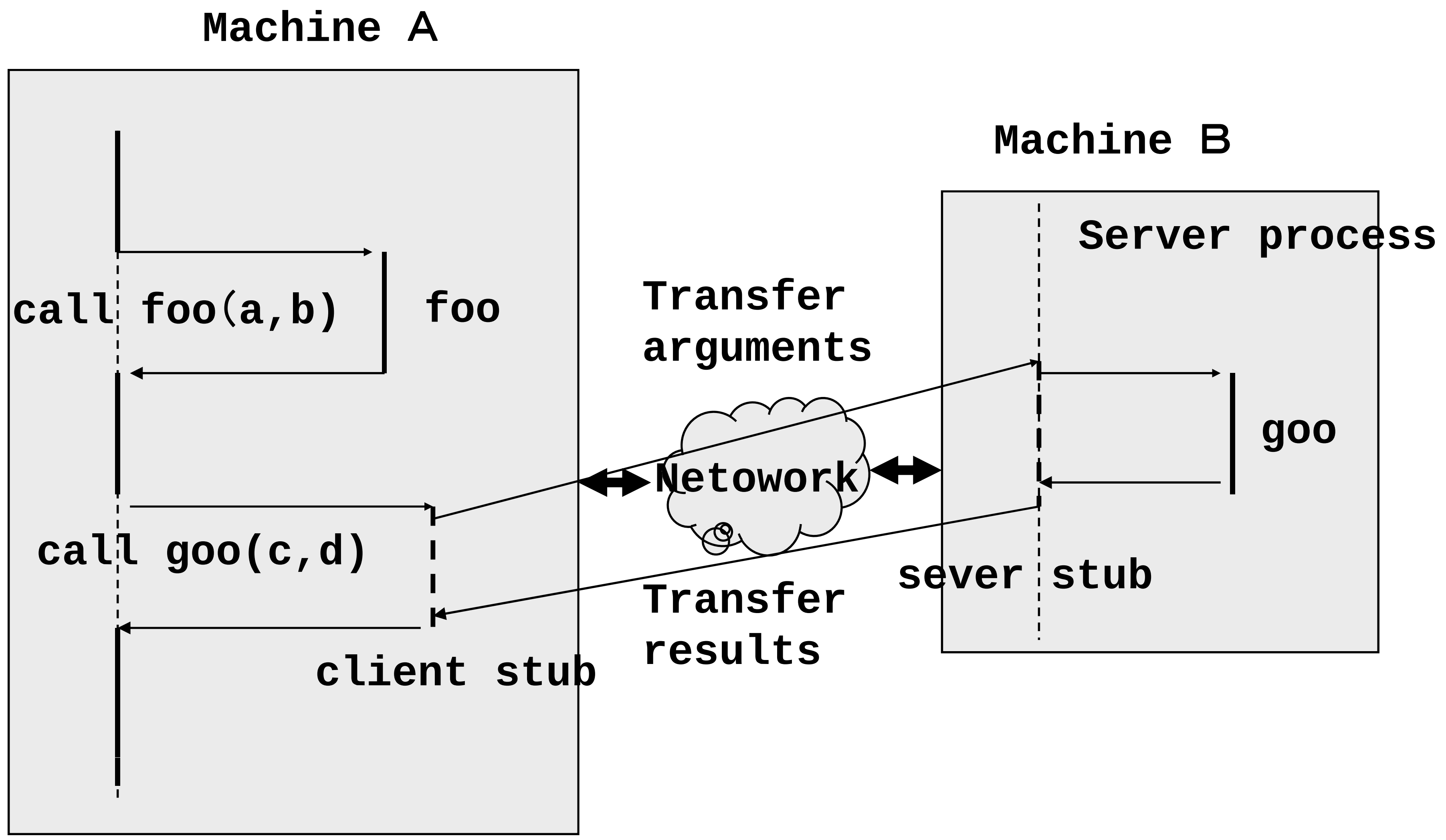
Client side(2)

```
    memset(&sin, 0, sizeof(sin));
     sin.sin_family = AF_INET;
     bcopy(hp->h_addr,&sin.sin_addr.s_addr,hp->h_length);
     sin.sin_port = port;

     fd = socket(AF_INET, SOCK_STREAM, 0);
     if(fd < 0){
         perror("socket failed");
         exit(1);
     }

     r = connect(fd,(struct sockaddr *)&sin,sizeof(sin));
     if(r < 0){
         perror("connect failed");
         exit(1);
     }

    sprintf(buf,"hello world...¥n");
     write(fd,buf,strlen(buf)+1);

     close(fd);
     exit(0);
}
```

“Conventional” function call and RPC



A more close look at RPC



Steps in RPC

① Client calls the associated client stub as by conventional procedure call.

② client stub creates a message containing the information of arguments and request the operating system to send it.

③ The operating system send the message to remote operating system through the network.

④ The remote operating system receive the message and send it to the server stub.

⑤ The sever stub takes the information of the arguments from the message.

⑥ The server stub calls the target function with the arguments, and obtain the results.

⑦ The server stub creates a messages containing the results and request the local operating system to send it.

⑧ The server's operating system send the message through the network.

⑨ The client operating system receive the massage to send it to the client stub.

⑩ The client stub take the information of the results and send it to the caller.

RPC Middleware and IDL

- **It is a time-consuming work to implement stubs for each remote functions.**

- **RPC middleware is a software to generate client stubs and server stubs from a description of functions interface (type of arguments, etc) in IDL(Interface Description Language).**
  - **SUN RPC**
  - **Java RMI**

- **By using this framework, the programmer can call the remote functions as the local function call.**

Other comments on RPC

- Asynchronous RPC
  - A remote procedure call which returns without waiting the completion of RPC.
  - RPC itself takes some times.
  - During execution of RPC, the client may wait idle until the RPC is returns. That means the client can do other useful work until the RPC returns.
- Naming service
  - When building distributed applications using RPCs, it is sometimes useful to maintain the database where the functions are available.
  - The name service is a database service to provide information where the functions are available.

Basic network programming in Java

Server side:

```
ServerSocket ss = new ServerSocket(port);
 Socket s = ss.accept();
 DataOutputStream out =
          new DataOutputStream(s.getOutputStream());
 out.writeInt(123);  /* write …*/
```

Client-side:

```
Socket s = new Socket(host, port);
DataInputStream in = new DataInputStream(s.getInputSteram());
y = in.readInt(); /* … read …*/
```

Object transfer in Java

- In Java, an object can be written/read into/from file/network by "Serialization" (the function to convert the object information as a text)
- Using this Serialization, an object can be transferred through network.

```
ObjectOutputStream out = new ObjectOutputStream(s.getOutputStream());
out.writeObject(obj);


ObjectInputStream in = new ObjectInputStream(s.getInputStream());
Object obj = in.readObject();
```

Interface definition of ShowData class

```
public interface ShowDate {
  public long getCurrentMillis();
  public long getMillis();
}
```

◆ Implemention of ShowDate Interface

```
public class ShowDateImpl implements Serializable, ShowDate {
  long millis = 0;
  Date date = null;
  public ShowDateImpl(){
    millis = getCurrentMillis();
    date = new Date(millis);
  }
  public long getCurrentMillis(){
    System.out.println("getCurrentMillis called!");
    return System.currentTimeMillis();
  }
  public long getMillis() {
    System.out.println("getMillis called!");
    return millis;
  }
  public static void main(String argv[]){
    ShowDateImpl sdi = new ShowDateImpl();
    System.out.println(sdi.date);
    System.out.println(sdi.getCurrentMillis());
  }
}
```

Server to transfer objects (sender)

```
public class ObjectServer {
  public static void main(String argv[]){
    try {
      int port = 8080;

      ServerSocket ss = new ServerSocket(port);
      while(true){
        Socket s = ss.accept();
        System.out.println("Object Server accept!!!");
        ObjectOutputStream oos =
          new ObjectOutputStream(s.getOutputStream());
        ShowDateImpl sd = new ShowDateImpl();
        System.out.println("write "+sd);
        oos.writeObject(sd);
        s.close();
      }
    } catch(Exception e){
      System.out.println("object write err:"+ e);
    }
  }
}
```

Server to transfer objects (receiver)

```
public class client0 {
  public static void main(String argv[]){
    try {
      client1 cl = new client1();
      String host = "localhost";
      int port = 8080;

      Socket s = new Socket(host,port);

      ObjectInputStream ois =
                   new ObjectInputStream(s.getInputStream());
      ShowDate sd = (ShowDate)(ois.readObject());
      System.out.println(sd.getCurrentMillis());
      System.out.println(sd.getMillis());
      System.out.println(sd);
    } catch(Exception e){
      System.out.println(e);
    }
  }
}
```

Network Class Loader (1)

```
public class NetworkClassLoader extends ClassLoader {
  InputStream in;
  ByteArrayOutputStream out = new ByteArrayOutputStream(1024);
  public NetworkClassLoader() {
    this("localhost",8081);
  }
  public NetworkClassLoader(String host, int port){
    try {
      Socket s = new Socket(host,port);
      in = s.getInputStream();
    } catch(Throwable e){
      System.err.print("cannot open socket");
      System.exit(1);
    }
  }
  protected class findClass(String name)
              throws ClassNotFoundException {
…
}
```

Network Class Loader (2)

```
  protected Class findClass(String name)
                   throws ClassNotFoundException {
     try {
       byte buff[] = new byte[1024];
       int n,m;
       int len = 0;
       while((n = in.read(buff,0,1024)) > 0){
         out.write(buff,0,n);
         len += n;
       }
       byte data[] = new byte[len];
       data = out.toByteArray();
       return defineClass(null,data,0,len);
     } catch(Throwable e){
       System.err.println("read err");
       throw new ClassNotFoundException();
     }
   }
 }
```

Class Sever with serves only ShowDataImpl class

```
public class ClassServer {
  public static void main(String argv[]){
    try {
      String classFile = "ShowDateImpl.class";
      int port = 8081;
      ServerSocket ss = new ServerSocket(port);
      while(true){
         Socket s = ss.accept();
         System.out.println("Class Server accept!!!");
         BufferedOutputStream bos =
           new BufferedOutputStream(s.getOutputStream());
         BufferedInputStream bis =
           new BufferedInputStream(new FileInputStream(classFile));
         int len;
         byte buff[] = new byte[256];
         while((len = bis.read(buff,0,256)) >= 0){
           bos.write(buff,0,len);
         }
         bos.flush();
         bos.close();
         bis.close();
      }
    } catch(Exception e){
      System.out.println("class file err:"+ e);
    }
```

Program to load class information using network class loader

```
public class client {
  public static void main(String argv[]){
    try {
      NetworkClassLoader loader = new NetworkClassLoader();
      Class cl = loader.loadClass("ShowDateImpl");
      ShowDate sd = (ShowDate)(cl.newInstance());
      System.out.println(sd.getCurrentMillis());
      System.out.println(sd);
    } catch(Exception e){
      System.out.println(e);
    }
  }
}
```

Object class loader with ObjectStream (1)

```
public class client1 {
  public static void main(String argv[]){
    try {
      client1 cl = new client1();
      String host = "localhost";
      int port = 8080;

      Socket s = new Socket(host,port);

      MyObjectInputStream ois =
        cl. new MyObjectInputStream(s.getInputStream(),
                                          new NetworkClassLoader());
      ShowDate sd = (ShowDate)(ois.readObject());
      System.out.println(sd.getCurrentMillis());
      System.out.println(sd.getMillis());
      System.out.println(sd);
    } catch(Exception e){
      System.out.println(e);
    }
  }
```

Object class loader with ObjectStream (2)

```
public class MyObjectInputStream extends ObjectInputStream {
    public ClassLoader cl;
    public MyObjectInputStream(InputStream im, ClassLoader cl)
         throws IOException {
           super(im);
           this.cl = cl;
    }
    protected Class resolveClass(ObjectStreamClass v)
                                   throws IOException {
      try {
        return super.resolveClass(v);
      } catch(ClassNotFoundException e){
        try {
          return cl.loadClass("ShowDateImpl");
        } catch(Exception e2){
          System.out.println(e2);
        }
      }
      return null;
    }
  }
}
```

Client using MarshalInputStream

```
public class client0 {
  public static void main(String argv[]){
    try {
      String host = "localhost";
      int port = 8080;
      if(System.getSecurityManager() == null){
               System.setSecurityManager(new RMISecurityManager());
      }
      System.out.println("security done...");
      Socket s = new Socket(host,port);
      System.out.println("socket="+s);
      MarshalInputStream ois =
              new MarshalInputStream(s.getInputStream());
      ShowDate sd = (ShowDate)(ois.readObject());
      System.out.println(sd.getCurrentMillis());
      System.out.println(sd.getMillis());
      System.out.println(sd);
    } catch(Exception e){
      System.out.println(e);
    }
  }
}
```

Sever using MarshalOutputStream

```
public class ObjectServer {
  public static void main(String argv[]){
    try {
      int port = 8080;

      if(System.getSecurityManager() == null){
         System.setSecurityManager(new RMISecurityManager());
      }
      System.out.println("security manager done ...");

      ServerSocket ss = new ServerSocket(port);
      System.out.println("accept ..."+ss);
      while(true){
         Socket s = ss.accept();
         System.out.println("Object Server accept!!!");
         MarshalOutputStream oos =
           new MarshalOutputStream(s.getOutputStream());
         ShowDateImpl sd = new ShowDateImpl();
         System.out.println("write "+sd);
         oos.writeObject(sd);
         s.close();
      }
    } catch(Exception e){
      System.out.println("object write err:"+ e);
    }
```

Client using MarshalObject

```
public class client {
  public static void main(String argv[]){
    try {
      String host = "localhost";
      int port = 8080;
      if(System.getSecurityManager() == null){
        System.setSecurityManager(new RMISecurityManager());
      }
      client cl = new client();
      Socket s = new Socket(host,port);
      ObjectInputStream ois =
                  new ObjectInputStream(s.getInputStream());
      MarshalledObject mo = (MarshalledObject)ois.readObject();
      System.out.println("Marshalled Object ="+mo);
      System.out.println("                Object ="+mo.get());
      ShowDate sd = (ShowDate)(mo.get());
      System.out.println(sd.getCurrentMillis());
      System.out.println(sd.getMillis());
      System.out.println(sd);
    } catch(Exception e){
      System.out.println(e);
    }
```

Server example using MarshalObject

```
public class ObjectServer {
  public static void main(String argv[]){
    try {
      int port = 8080;
      if(System.getSecurityManager() == null){
        System.setSecurityManager(new RMISecurityManager());
      }
       ObjectServer os = new ObjectServer();
      ServerSocket ss = new ServerSocket(port);
      System.out.println("accept ..."+ss);
      while(true){
        Socket s = ss.accept();
        System.out.println("Object Server accept!!!");
        ObjectOutputStream oos =
          new ObjectOutputStream(s.getOutputStream());
        ShowDateImpl sd = new ShowDateImpl();
        System.out.println("write "+sd);
        oos.writeObject(new MarshalledObject(sd));
        s.close();
      }
    } catch(Exception e){
      System.out.println("object write err:"+ e);
```

Summary: Transferring Objects through network

- In order to reference (compile) an object, only the interface must be shared. The interface is defined as a Java interface definition. The user does not need the implementation of classes and its location.

- The Java object transfer mechanism, ObjectStream Class, sends only class name and data of the object. Thus, in order to execute any methods o f the transferred object, the program of the implementation is required where the method is executed.

- To transfer the code, the mechanism to transfer the class file is required. Usually, http server is used to transfer class files. One of such systems is MarshalledObjectStream. The URL of the http server is specified by rmi.server.codebase.

Overview of RMI

- Java interface definition is used as IDL descriptions. Both client and Server have to have the interface definition of the shared objects.
- The interface definition of the shared object must be extended from “Remote” interface to generate stubs. (The shared object is a subcalss of “remote”).
- Use “rmic” (RMI compiler) to generate stubs from the interface definition extended by ”Remote” interface. The generated class file contains Skelton xxx_skel.class and Stub xxx_stub.class. Place these generated class file in the class path of the network class loader.
- In server-side, run “rmiregisty” which manages remote objects. Rmiregistry provides naming service in RMI.
- In the implementation of the shared class, it is defined with UnicastRemoteObject as a superclass, and register it as a remote object.
- On client-side, retrieve the registered remote object in the server, and call a method of the remote object.

Steps to transfer data in RMI

- Before the executing program, start up the network class server (it can be http sever) (http://localhost:8081)
- Zip the shared class files to Jar file (dl.jar) , placed under the network class sever.
- When starting the client side program, specify where the class file is downloaded. (codebase).
- Both program have to set up security manager and security policy.

```
java –Djava.rmi.sever.codebase=http://localhost:8081/dl.jar
     –Djava.security.policy=policy ObjectSever
```

Interface Definition of RMI Remote object

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface ShowDate extends Remote {
  public long getCurrentMillis() throws RemoteException;
  public long getMillis() throws RemoteException;
}
```

Implementation and registration of RMI remote object (1)

```
public class ShowDateImpl extends
        UnicastRemoteObject implements ShowDate {
  long millis = 0;
  Date date = null;
  public ShowDateImpl() throws RemoteException {
    super();
    millis = getCurrentMillis();
    date = new Date(millis);
  }
  public long getCurrentMillis() throws RemoteException {
    System.out.println("getCurrentMillis called!");
    return System.currentTimeMillis();
  }
  public long getMillis() throws RemoteException {
    System.out.println("getMillis called!");
    return millis;
  }
  public static void main(String argv[]){
    if(System.getSecurityManager() == null){
      System.setSecurityManager(new RMISecurityManager());
    }
```

Implementation and registration of RMI remote object (2)

```
public class ShowDateImpl extends
       UnicastRemoteObject implements ShowDate {
  ……
  public static void main(String argv[]){
    if(System.getSecurityManager() == null){
      System.setSecurityManager(new RMISecurityManager());
    }
    try {
      ShowDateImpl sdi = new ShowDateImpl();
      Naming.rebind("//localhost/TimeServer",sdi);
      System.out.println("TimeServer bound in registry");
    } catch(Exception e){
      System.out.println(e.getMessage());
    }
  }
}
```

Client using RMI remote objects

```
public class client {
  public static void main(String argv[]){
    if(System.getSecurityManager() == null){
      System.setSecurityManager(new RMISecurityManager());
    }
    ShowDate obj = null;
    try {
      String location = "rmi://localhost/TimeServer";
      obj = (ShowDate)Naming.lookup(location);

      long remote_millis = obj.getCurrentMillis();
      long local_millis = System.currentTimeMillis();
      System.out.println("remote =" + remote_millis);
      System.out.println("local =" + local_millis);
    } catch(Exception e){
      System.out.println(e);
    }
  }
}
```

Activation

- Actutally, the remote object program of RMI does not terminate. As far as the object is managed by rmiregistry, rmregistry keeps reference to the remote RMI object. That’s, the remote object is waiting for requests.
- In this implementation, the remote object must be stand by alive a long time (forever?). If the number of objects is getting large, many process have to be invoked and stand-by, resulting in inefficiency.
- Activation is a mechanism to keep the remote object in in-active state (stored in the file system by serialization), and to re-invoke these in-active remote object by a demon rmid upon a request. It is introduce from JDK 1.2.
- To use activation, use class java.rmi.activation.Activatable in stead of UnicastRemoteObject.

Example using RMI activation

```
public class ShowDateImpl extends Activatable implements ShowDate {
  public static void main(String argv[]){
    if(System.getSecurityManager() == null){
      System.setSecurityManager(new RMISecurityManager());
    }
    try {
      Properties props = new Properties();
      props.put("java.security.policy",
                 "/home/msato/java/tmp/rm-test5/serv/policy.txt");
      ActivationGroupDesc myGroup =
                       new ActivationGroupDesc(props,null);
      ActivationGroupID agi =
        ActivationGroup.getSystem().registerGroup(myGroup);
      ActivationGroup.createGroup(agi,myGroup,0);
      String location = "file:/home/msato/java/tmp/rm-test5/serv/";
      ActivationDesc desc =
              new ActivationDesc("ShowDateImpl",location,null);
      ShowDate rmi = (ShowDate)Activatable.register(desc);
      System.out.println("Got the stub for the ShowDateImpl = "+rmi);
      Naming.rebind("//localhost/TimeServer",rmi);
      System.out.println("Exported ShowDateImpl...");

      System.exit(0);
    } catch(Exception e){
      System.out.println(e.getMessage());
```

Jini

- Jini is a network service architecture for the construction of distributed systems in the form of modular co-operating services, proposed by Sun micro systems.
  - Jini was transferred to Apache River project.
- Jini technology can be used to build adaptive network systems that are scalable, evolvable and flexible as typically required in dynamic computing environments. Jini offers a number of powerful capabilities such as service discovery and mobile code.
- Jini proposes several mechanisms on distributed object programming in order to “federate” several kinds of computers including micro-chip in home appliances (TV, washing machine …), server, to supercompter.

Jini's lookup Service

- An interesting and important concept of Jini is "service"
  - Computer connected to the network is not just a computer, but also object to give some service.
  - Distributed systems works to exchange services with each others.

- Service can be provided anywhere in the network.
- The first step in creating a Jini service is for the service to find the lookup service (LUS) - a process called discovery
- By using LUS, any object can lookup (discover) a service and use it.
- This idea is similar to "DHPC …"

- RMI provides registry mechanism which manages and maintains the objects in the server. Jini Loopup services is more distributed an extended to network-wide.

Question and Answer