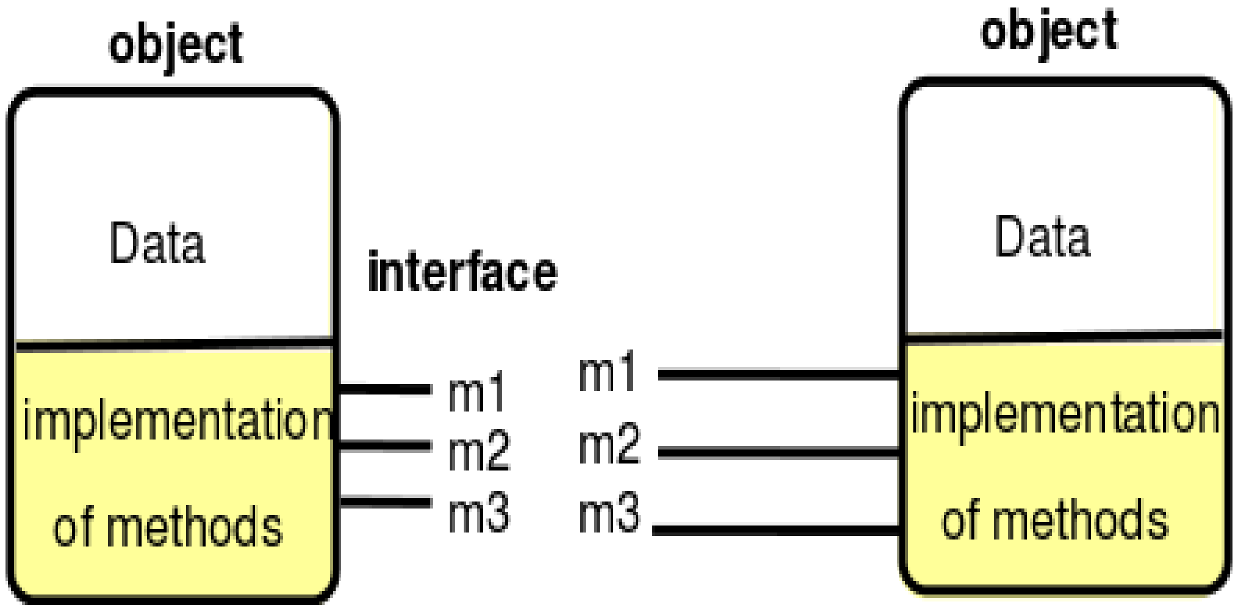
JAVA RMI

DISTRIBUTED OBJECTS and RMI



Java Remote interfaces *Shape* and *ShapeList*

```
import java.rmi.*;
import java.util.Vector;
public interface Shape extends Remote {
    int getVersion() throws RemoteException;
    GraphicalObject getAllState() throws RemoteException;
}
public interface ShapeList extends Remote {
    Shape newShape(GraphicalObject g) throws RemoteException;
    Vector allShapes() throws RemoteException;
    int getVersion() throws RemoteException;
}
```

Parameter and result passing

Parameters of the methods -input parameters

result of the method – output parameter

- Serializable object -argument or result of RMI

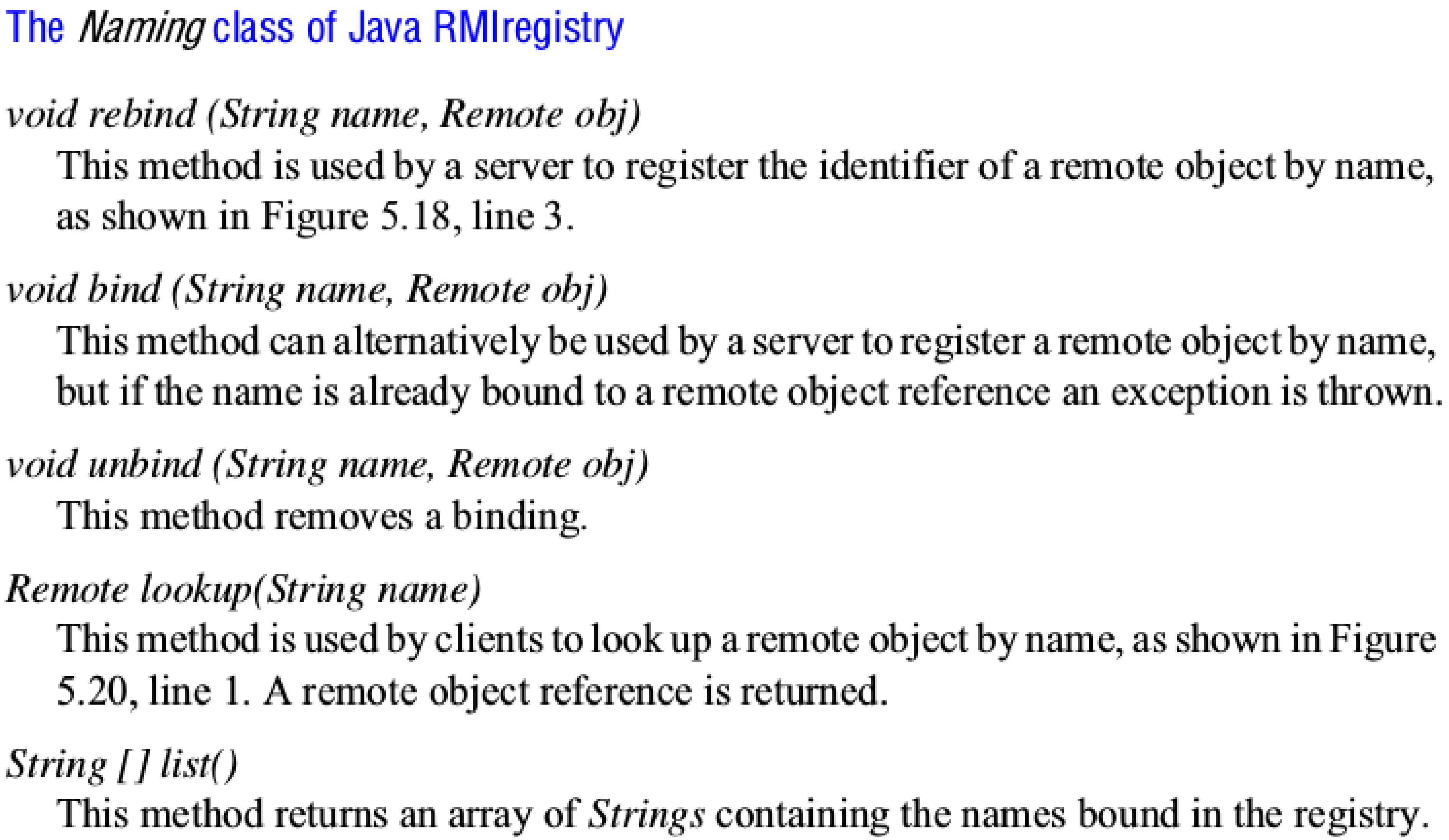
- Passing remote objects
- Passing non remote objects

RMI Registry

- The RMIregistry is the binder for Java RMI.
- An instance of RMIregistry should normally run on every server computer that hosts remote objects.
- It maintains a table mapping textual, URL-style names to references to remote objects hosted on that computer.
- It is accessed by methods of the Naming class, whose methods take as an argument a URL-formatted string of the form:

  computerName:port/objectName
- where computerName and port refer to the location of the RMIregistry.

NAMING CLASS OF RMI REGISTRY



The *Naming* class of Java RMIregistry

void rebind (String name, Remote obj)
This method is used by a server to register the identifier of a remote object by name, as shown in Figure 5.18, line 3.

void bind (String name, Remote obj)
This method can alternatively be used by a server to register a remote object by name, but if the name is already bound to a remote object reference an exception is thrown.

void unbind (String name, Remote obj)
This method removes a binding.

Remote lookup(String name)
This method is used by clients to look up a remote object by name, as shown in Figure 5.20, line 1. A remote object reference is returned.

String [] list()
This method returns an array of *Strings* containing the names bound in the registry.

Binding client and server programs

Java class *ShapeListServer* with *main* method

```
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
public class ShapeListServer{
    public static void main(String args[]){
        System.setSecurityManager(new RMISecurityManager());
        try{
            ShapeList aShapeList = new ShapeListServant();                1
            ShapeList stub =                                                2
                (ShapeList) UnicastRemoteObject.exportObject(aShapeList,0);3
            Naming.rebind("//bruno.ShapeList", stub );                      4
            System.out.println("ShapeList server ready");
        }catch(Exception e) {
            System.out.println("ShapeList server main " + e.getMessage());}
    }
}
```

Java class *ShapeListServant* implements interface *ShapeList*

```
import java.util.Vector;
public class ShapeListServant implements ShapeList {
        private Vector theList;                  // contains the list of Shapes
        private int version;
        public ShapeListServant(){...}
        public Shape newShape(GraphicalObject g) {
                version++;
                Shape s = new ShapeServant( g, version);
                theList.addElement(s);
                return s;
        }
        public  Vector allShapes(){...}
        public int getVersion() { ... }
}
```

Java client of *ShapeList*

```
import java.rmi.*;
import java.rmi.server.*;
import java.util.Vector;
public class ShapeListClient{
    public static void main(String args[]){
        System.setSecurityManager(new RMISecurityManager());
        ShapeList aShapeList = null;
        try{
            aShapeList  = (ShapeList) Naming.lookup("//bruno.ShapeList");
            Vector sList = aShapeList.allShapes();
        } catch(RemoteException e) {System.out.println(e.getMessage());
        }catch(Exception e) {System.out.println("Client: " + e.getMessage());}
    }
}
```

Callback

- instead of clients polling the server to find out whether some event has occurred, the server should inform its clients whenever that event occurs.
- The term callback is used to refer to a server’s action of notifying clients about an event.