Poseidon House
Castle Park
Cambridge CB3 0RD
United Kingdom

TELEPHONE: **Cambridge (01223) 515010**
INTERNATIONAL: **+44 1223 515010**
FAX: **+44 1223 359779**
E-MAIL: **apm@ansa.co.uk**

Training

ANSAwise - Persistent Data Storage with CORBA

Chris Mayers

Abstract

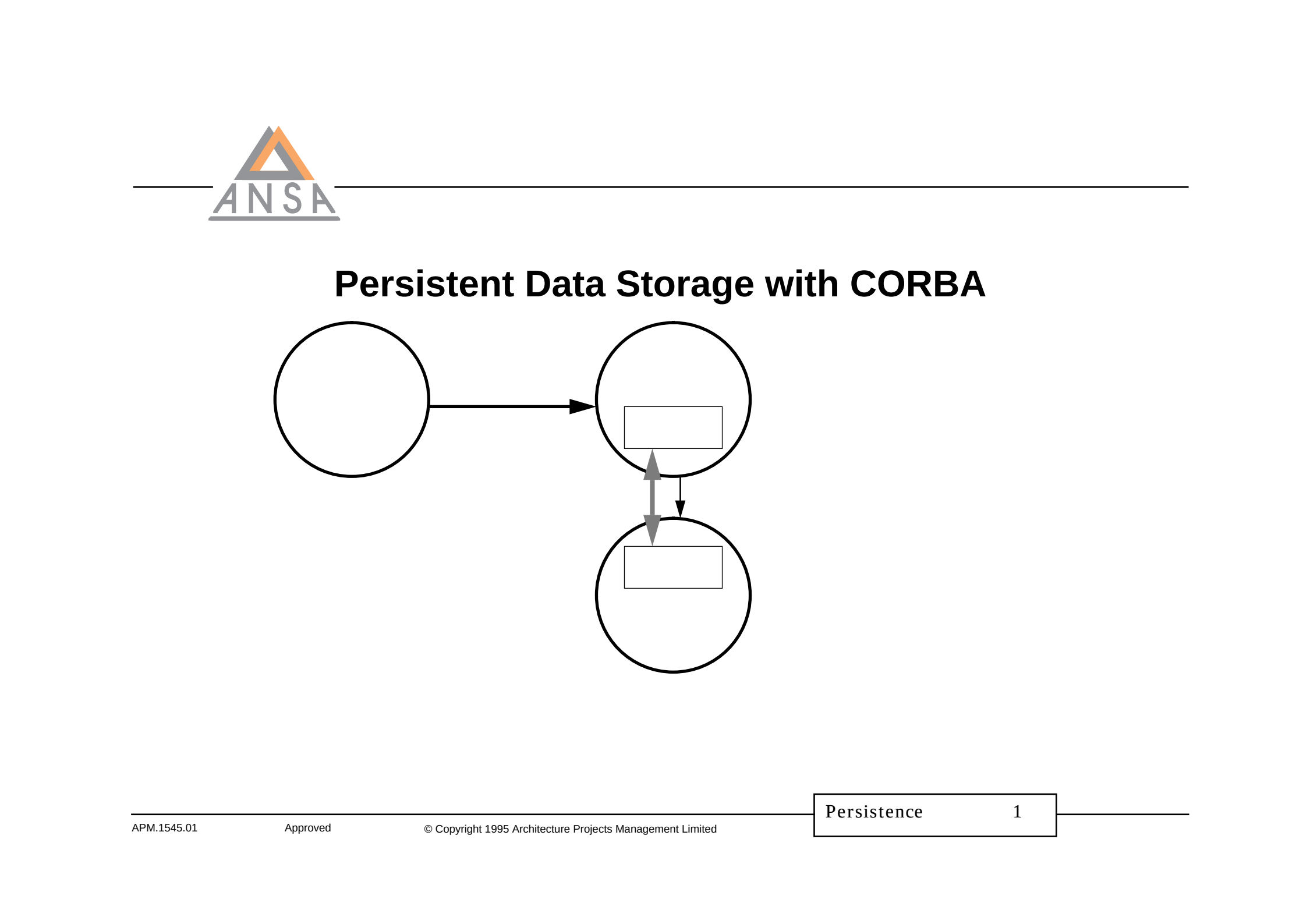
Almost all applications require persistent data storage facilities. Sometimes this may be achieved using a database; at the other extreme, it could be implemented in non-volatile RAM. The interfaces to these persistent data storage facilities are usually very different. This diversity of interfaces is problematic if applications portability is needed across different data storage facilities.

This module of the ANSAwise training programme describes the CORBA Persistent Object service and Externalization service.

[This module could also draw on the insights of the ISA work on the ANSA Storage Model, but this is now some years old. It could also mention the DIOMEDES real-time database used in the MOTOS project.]

APM.1545.01 **Approved** 31st July 1995
Briefing Note

Distribution:
Supersedes:
Superseded by:



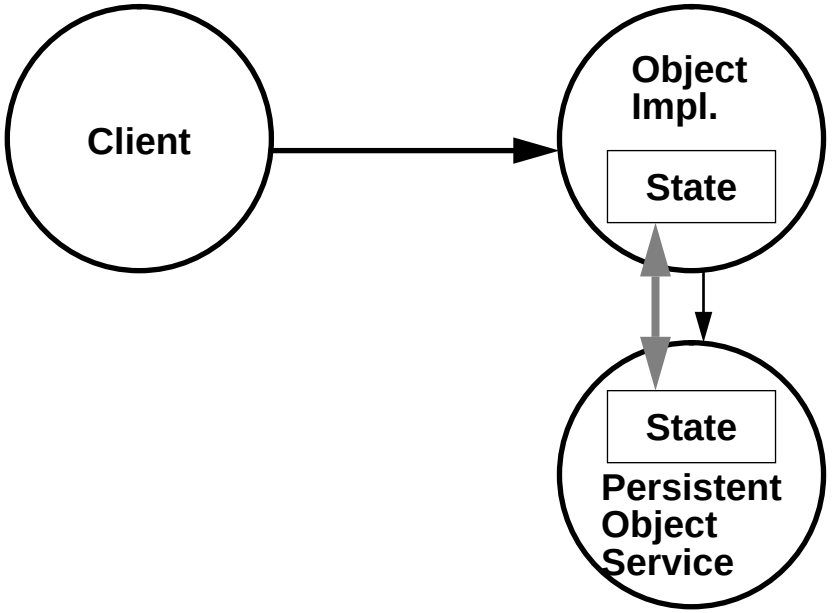
Persistent Data Storage with CORBA

In this session

- ***Examine how data storage is supported in CORBA***
  - **in the Persistent Object (persistence) service**
  - **in the Externalization service**

Roles in the Persistent Object Service

- ***Persistent Object Service stores persistent state***



Persistent state

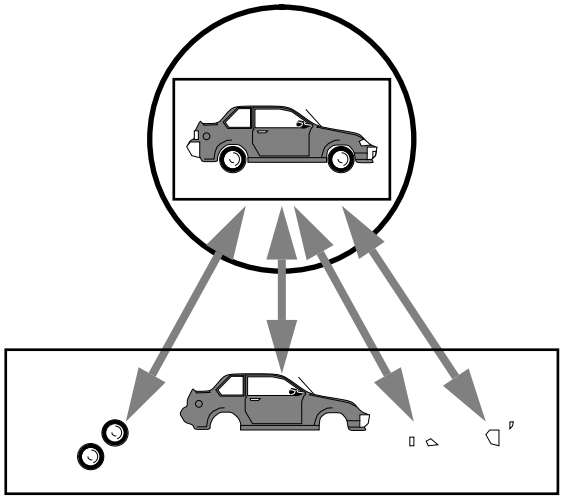
- *The state of an object can be treated as two parts*
- *Dynamic state*
  - typically in memory
  - lost if the object crashes
- *Persistent state*
  - typically on disk
  - preserved over crashes, and can be used to reconstruct the dynamic state

Preserving state

- ***It is an object's responsibility to preserve its dynamic state as persistent state***
  - **and to recover it after a crash**
- ***The object may use the Persistent Object service for this purpose***
  - **or any private mechanism it chooses instead...**
  - **... flat files**
  - **... direct access to a relational database**
  - **... direct access to some other kind of database**
- ***The object may delegate the responsibility back to its client***

Advantages of using the Persistent Object Service

- *Typical data storage mechanisms do not have object characteristics*
  - uniform interfaces, self-description, and abstraction



- *This is sometimes known as an 'impedance mismatch'*

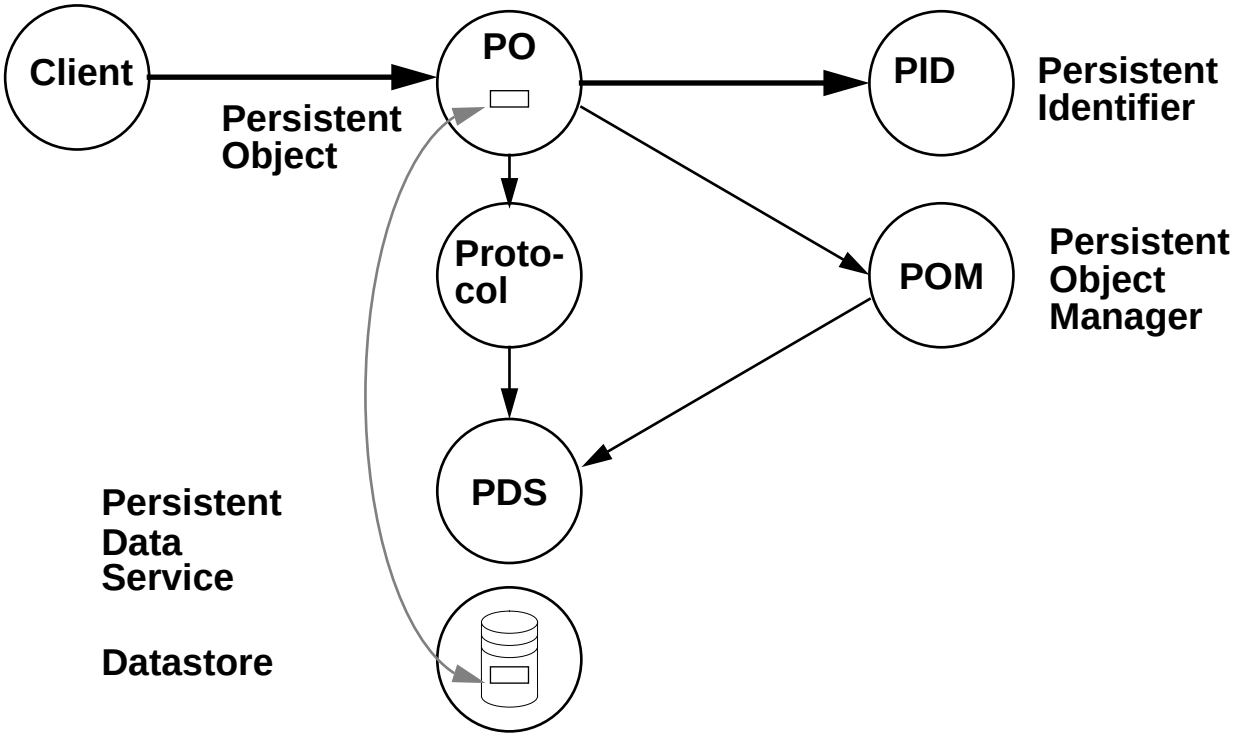
Client Control

- *Clients may need to control or manage persistence of the objects they use, in particular*
- *In particular, they client may need to control*
  - exactly when persistent state is saved and restored
  - which copy of persistent state is to be used
- *Object implementations do not provide client control*
  - they may choose to hide the complexity from the client

Persistent Object Service Implementations

- ***As with all CORBA Object Services, a wide range of implementations is envisaged***
  - **on top of relational databases**
  - **on top of object-oriented databases**
  - **on top of specialized compound document storage mechanisms**
  - **on top of lightweight storage mechanisms**
  - **... or any combination of these**

Major Components of the Persistent Object Service



Persistence Components - 1

- ***Persistence Identifier (PID)***
  - **describes the location of an object's data in some Datastore**
- ***Persistent Object (PO)***
  - **an object whose persistence is controlled externally (by some client)**
- ***Persistent Object Manager (POM)***
  - **supports both internally and externally controlled persistence**
- ***Persistent Data Service***
  - **coordinates basic persistence operations between Datastore and Protocol**

Persistence Components - 2

- ***Protocol***
  - encapsulates a general kind of data storage
- ***Datastore***
  - basic access to the underlying data storage mechanisms

Persistent Identifier (PID)

```
module CosPersistencePID {

   interface PID {
      attribute string datastore_type;
      string get_PIDstring();
   };

};
```

- *An object requires a PID to store data persistently*
- *Do not confuse a PID with an object reference*
  - object reference identifies an object
  - persistent identifier identifies an object's state

Specialized PIDs

- *There also may be specialized PIDs*
  - with additional operations and attributes...
  - ... with specifications inherited from PID
- *For example*

```
interface PID_DB : CosPersistencePID::PID {
   attribute string database_name;
};

interface PID_SQLDB : PID {
   attribute string sql_statement;
};
```

PID Factory example

```
interface PIDFactory {
   CosPersistencePID::PID create_PID_from_key(in string key);
   CosPersistencePID::PID create_PID_from_string (
      in string pid_string);
   CosPersistencePID::PID create_PID_from_string_and_key (
      in string pid_string, in string key);
};
```

- *To create a PID, you need to identify a PID implementation to use*
  - identified by key, string, or both
- *Supplying both may be useful when moving persistent data between Datastores with different interfaces*
- *This specification is only an example; others are also possible*

Persistent Object

- ***Has three interfaces***
  - **PO: for allowing a client to control persistence externally**
  - **POFactory: for creating POs**
  - **SD: for the persistent object to maintain internal consistency**

interface PO

```
interface PO {
   attribute COSPersistencePID:: PID p;
   COSPersistencePDS::PDS connect (
      in CosPersistencePID::PID p);
   void disconnect (in CosPersistencePID::PID p);
   void store (in CosPersistencePID::PID p);
   void restore (in CosPersistencePID::PID p);
   void delete (in CosPersistencePID::PID p);
};
```

- ***Allows a client to connect PO with a Datastore***
  - **when disconnected, the data in the PO and the Datastore are the same**
- ***Allows a client to move data between PO and Datastore in either direction***
  - **store/restore**

Synchronized Data

```
interface SD {
   void pre_store();
   void post_restore();
};
```

- ***Some objects have persistent and transient data***
  - **for example transient data in a cache, or redundant data structures**
- ***Operations in the Synchronized Data interface are only invoked by the POM***

Persistent Object Manager

- *Interface is very similar to the PO interface*
- *Externally-controlled POs invoke the POM when they are themselves invoked*
- *Internally-controlled POs invoke the POM according to their internal policy*

Persistent Data Service

- ***Interacts with the object to get data in and out of the object***
  - **using a Protocol**
- ***Interacts with the Datastore to get data in and out of the object***
- ***A PDS may support several Protocols***
- ***A PDS may use either a standard or a proprietary interface to its Datastore***

Protocols

- *The Persistent Object Service specifies three protocols*
  - Direct Access (PDS_DA) Protocol: uses attributes to store the data - the DDL is a subset of CORBA IDL, for example

```
interface MyDataObject {
   attribute short my_short;
   attribute float my_float;
};
```

  - ODMG-93 Protocol: similar to PDS_DA, but uses ODMG ODL (Object Definition Language)
  - Dynamic Data Object (DDO) Protocol: a Datastore-neutral protocol
- *Other protocols could be integrated*

Datastores

- *The Persistent Object Service specifies one Datastore*
  - Datastore_CLI: for record-oriented databases, based on the X/Open Data Management Call Level Interface
- *Other Datastores are not specified because*
  - other standard interfaces already exist
  - Protocols can drive their underlying database directly if they wish

Externalization

- ***Externalization is a separate Object Service***
- ***Externalization records an object's state in a stream of data***
  - **in memory, on disk, across the network**
- ***The stream owns the externalized form***
- ***The externalized form can be stored indefinitely and transported outside an ORB***
- ***The stream can be later internalized***
  - **into a new object...**
  - **...in a different ORB, not necessarily connected to the previous ORB**

Externalization Portability

- *There are many possible storage media and data formats*
- *To ensure data portability, the Externalization Service defines a standard external representation*
  - and an interface to externalize to a file

Externalization - the client's view

- *The client must first acquire an object reference for a stream*
  - possibly creating it via a StreamFactory
- *The client then invokes externalization for an object on a stream*
  - this may cause other objects to be externalized, but the client is unaware of this
- *A client later invokes internalization for an object from the same stream*
  - supplying a factory finder...
  - ... which will create a new object with the data from the stream

Externalization - the object's view

- ***Every object that wishes to be externalizable must support the Streamable interface***

- ***When the object receives an externalize_to_stream request it must write its state to the stream***

  - **using write_<type> operations for each of the CORBA basic types**

Externalization - the stream's view

- *The stream co-operates with the externalizable object*
  - using the object's Streamable interfaces
- *The stream supplies a StreamIO object to the externalizable object*
  - StreamIO actually writes the data to the externalized form

Externalizing Multiple Objects

- ***Multiple objects can be externalized to the same stream***
  - **by bracketing within a begin/end ‘context’ pair**
- ***This requires the objects to coordinate their externalization***
  - **for compound externalization of graphs of related objects**
- ***There is also specific support for externalizing relationships and roles***

Persistence, Externalization, and Lifecycle

- ***Saving and restoring data with the Persistent Object Service affects a single object***
- ***Externalizing and internalizing with the Externalization Service creates a new object***
- ***Copying an object with the LifeCycle Service creates a new object***

Summary

- ***The Persistent Object Service provides considerable flexibility***
  - **and supports both external (client) and internal control**
- ***The Externalization Service provides storage outside the ORB***
  - **and supports data portability between ORBs**
- ***For more information***
  - **about these services, see *CORBAservices* by the Object Management Group (Wiley)**
  - **about ODMG, see *The Object Database Standard: ODMG-93* by R. Cattell (Morgan Kaufmann)**