# Supplements on

# Object-Oriented Programming & Design

http://www.cs.sysu.edu.cn/~lwj/object-oriented/

**Dr. LI Wenjun**

lnslwj@mail.sysu.edu.cn
http://www.cs.sysu.edu.cn/~lwj/

**Department of Computer Science**

**SUN YAT-SEN UNIVERSITY, GZ 510275**

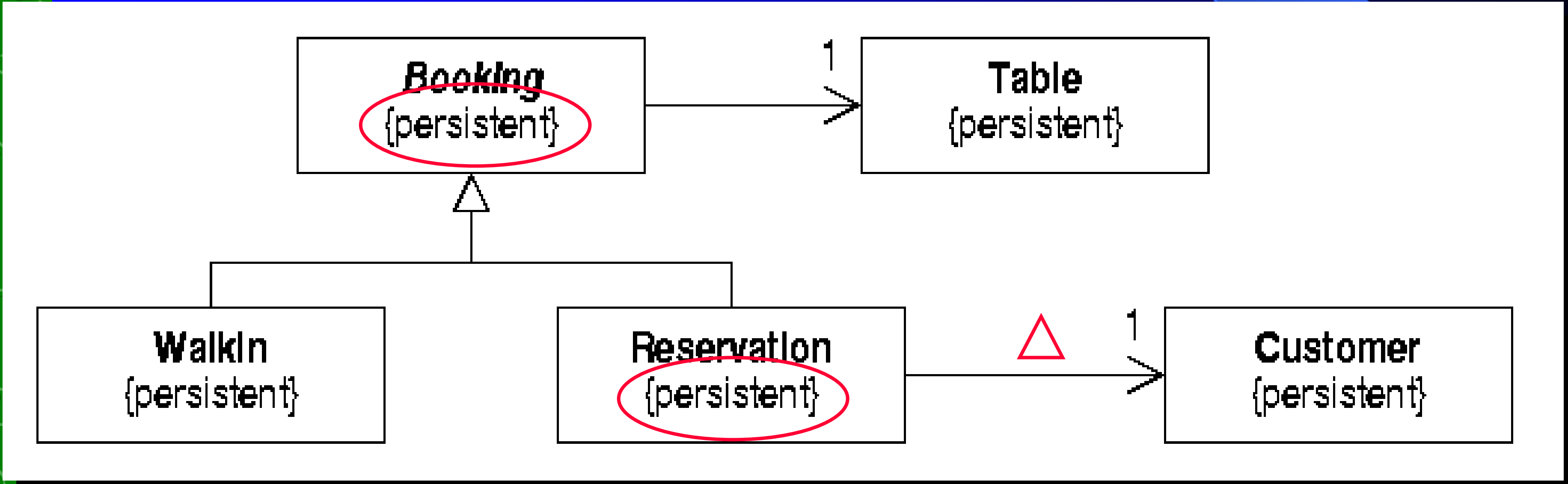
# Object Persistency Design

- Software Process
  - 主要工作流：**业务建模、分析、设计、实现**。
  - 持久性属于设计阶段的问题，而不是业务建模或分析的问题。
- Design Stage
  - 标识哪些数据需要持久性
  - 设计一个合适的数据库模式

# Identify Persistent Classes

- Basic Units in UML Designated with Persistency
  - 并不是每一个类都需要持久性。
    - **一个持久类中的属性也未必都需要实现持久性。**
  - 注意关联也需要考虑持久性问题。
    - **关联实际上也等价于类的属性。**

- Identify Persistency with *Tagged Values*
  - 通常采用简写的方式来表示（名字，值）。
    - 例如persistence=persistent简记为persistent
    - 默认值为transitory (或transient)

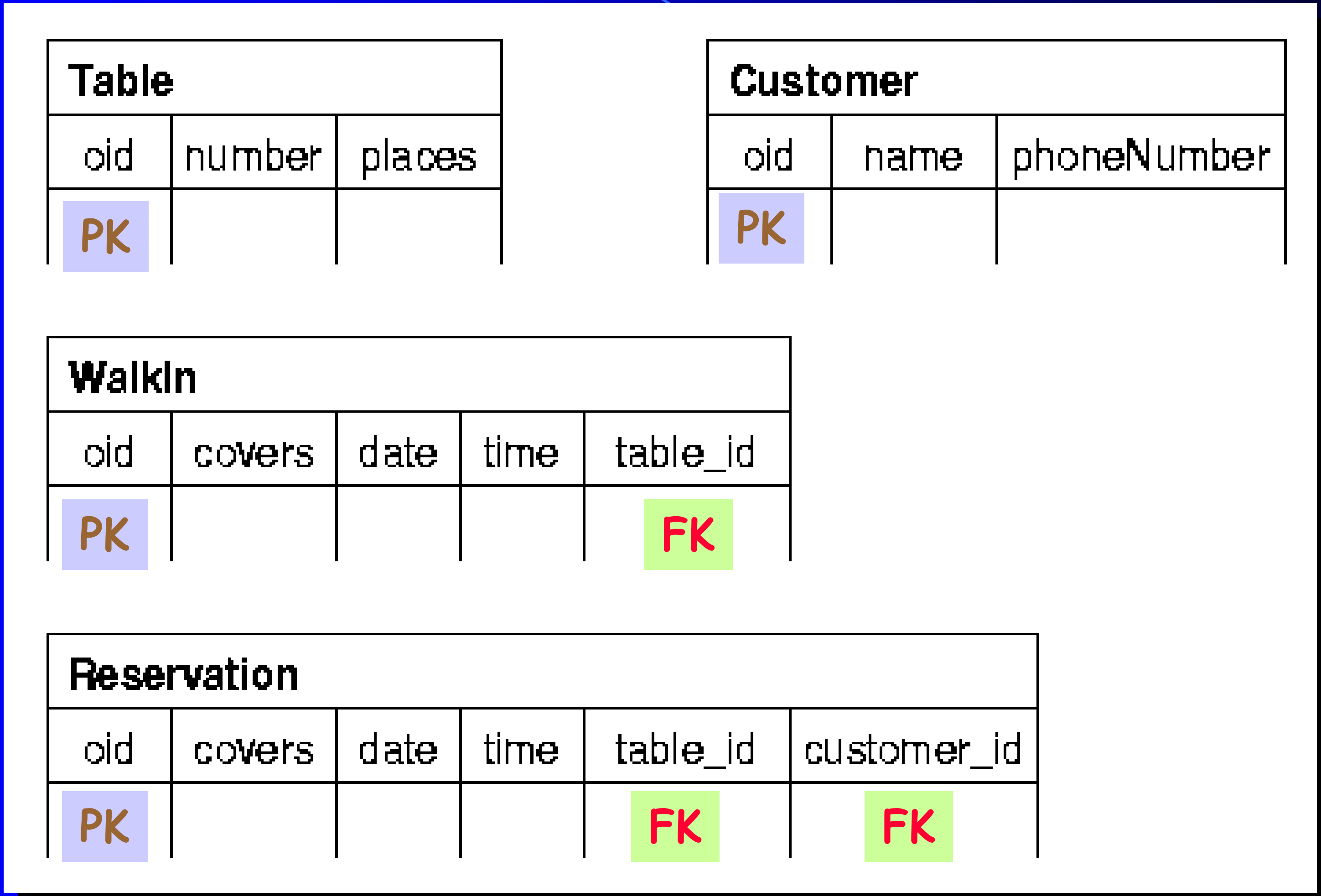


△ 两个持久类之间的关联也是持久的！

# Object/Relation Mapping

- **Classes** mapping to **Tables**
- **Associations** mapping to **Relations between Tables**
  - 为每一张表添加显式的对象标识（*oid*）。
  - 实现链接时用这些对象标识作为外码（*FK*）。
- **Generalization** has no direct mappings
  - 由于**`Booking`**是一个抽象类，可以简单地将其**具体子类**映射为**表**。

## Design of Database Schemes



**Table**

| oid | number | places |
|---|---|---|
| PK | | |

**Customer**

| oid | name | phoneNumber |
|---|---|---|
| PK | | |

**WalkIn**

| oid | covers | date | time | table_id |
|---|---|---|---|---|
| PK | | | | FK |

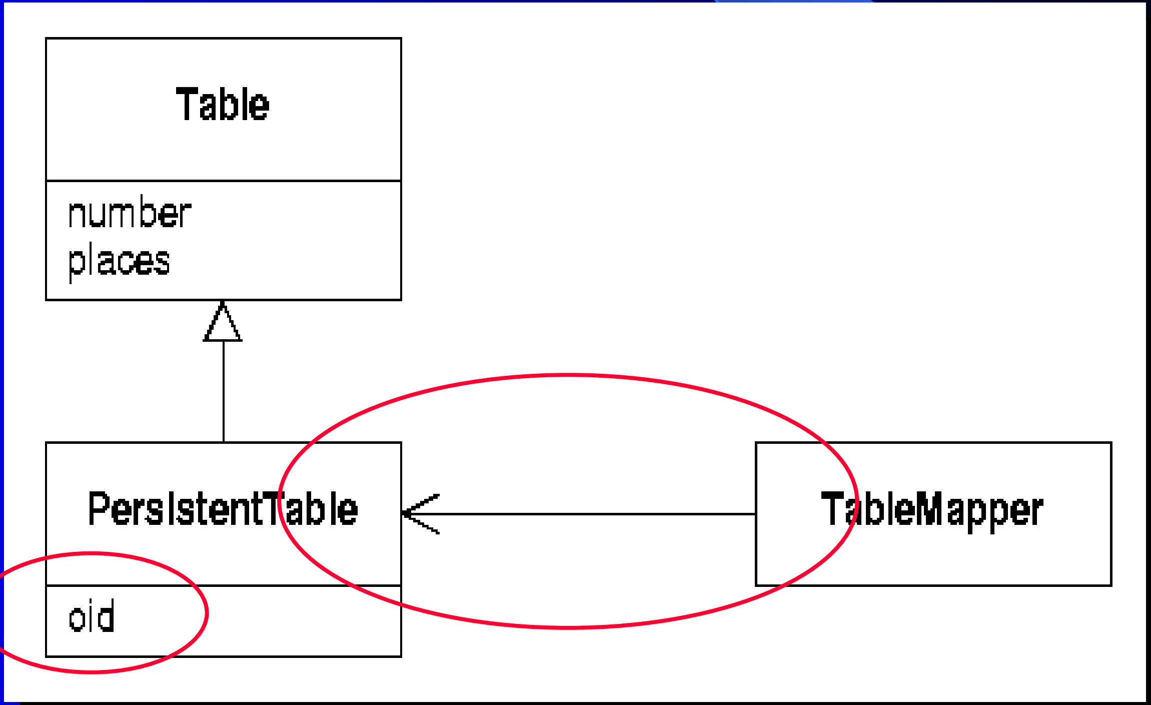
**Reservation**

| oid | covers | date | time | table_id | customer_id |
|---|---|---|---|---|---|
| PK | | | | FK | FK |

## Object/Relation Synchronization

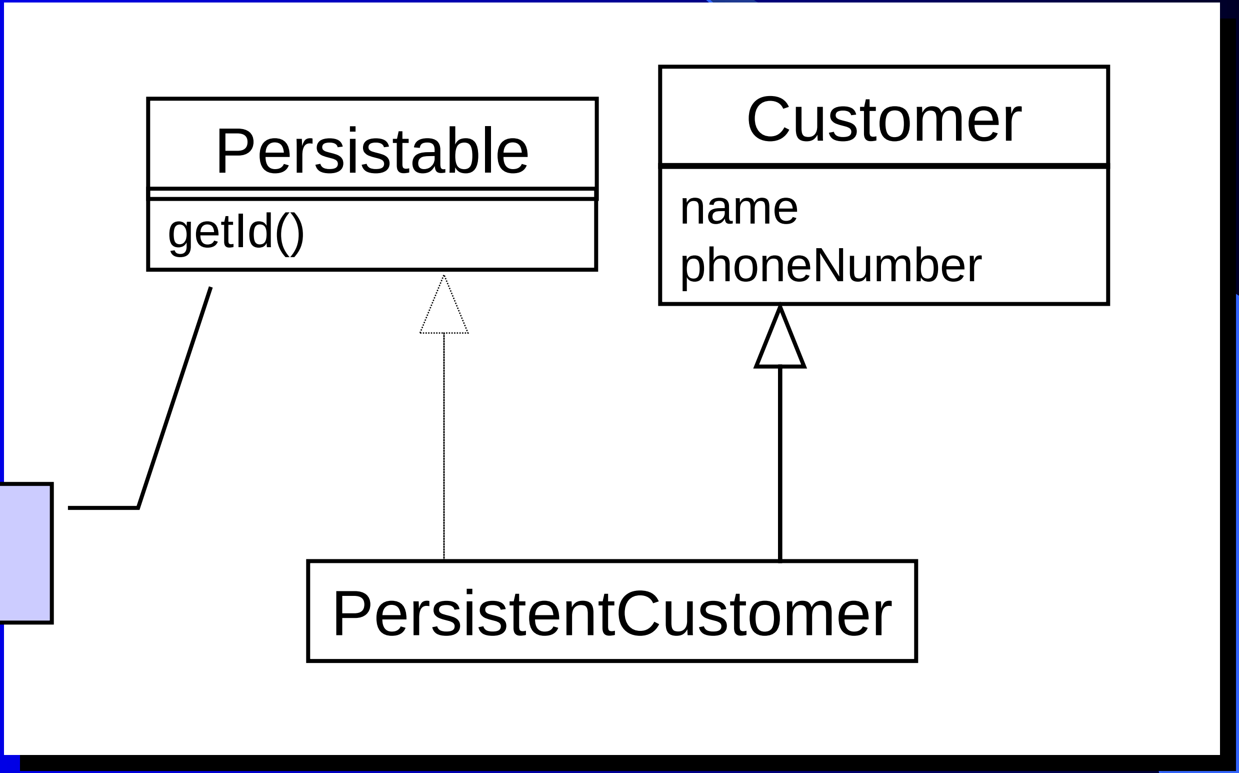
- Which object's responsibility should include saving or loading object states to or from database ?
  - 指派给已有的类会导致聚合性降低,
  - 故引入一个新的类专门执行该职责。
    - **为每一持久类定义一个映射类（Mapper）。**

- Introduce Object Ids into the Design Model
  - 从而在领域模型之外亦可实现持久性。
  - 为每一持久类定义子类，在子类中添加对象标识。
- **Mapper** class
  - Subclass handling persistency



## Another Approach to Introduction of Object Ids

– 系统中往往有多个持久对象：抽取持久性的共性。

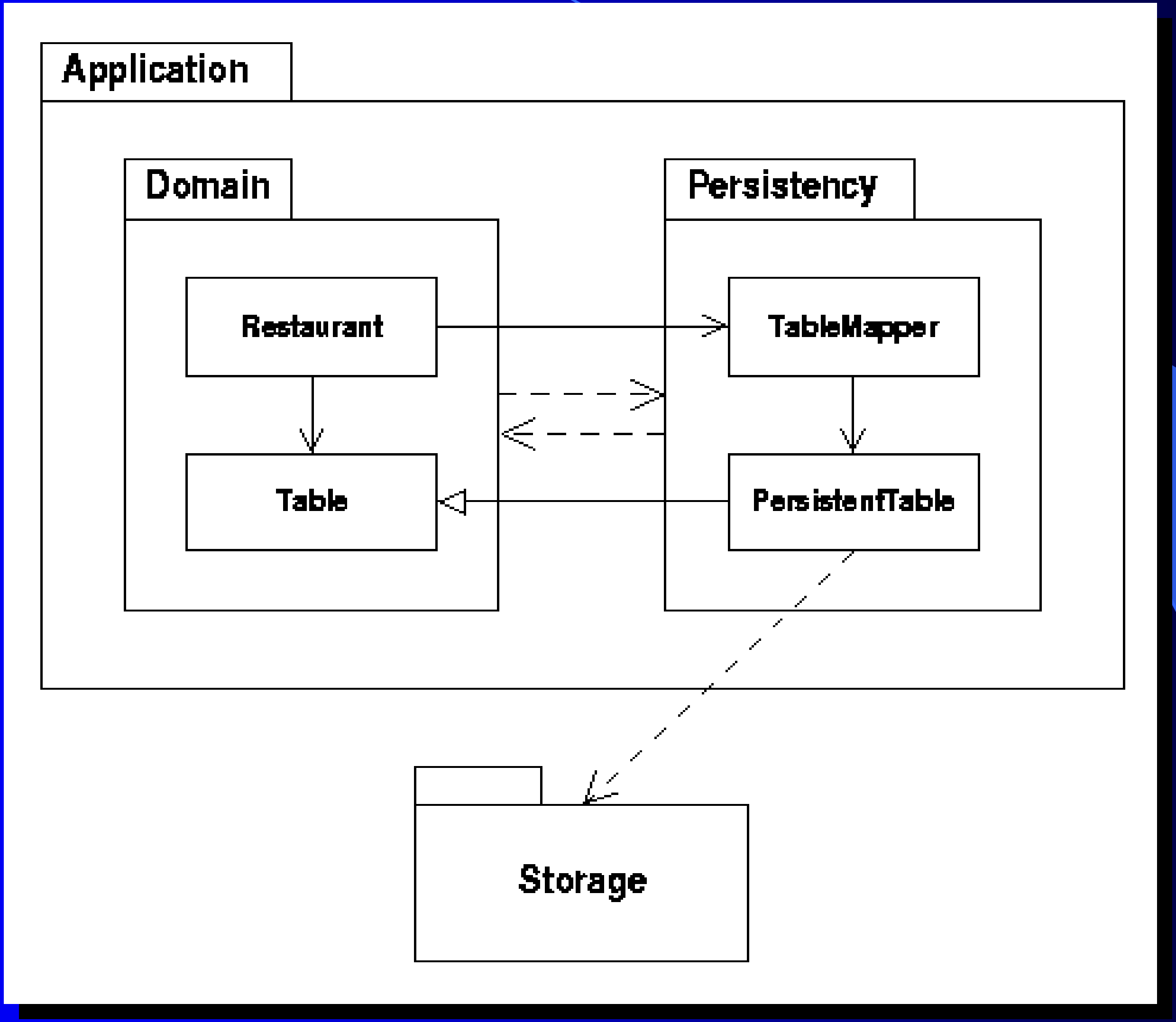


多继承：可改为抽象类，
引入：oid: Object;

## Persistency Architecture

- **`Persistent`** subclass and **`Mapper`** class depends their supporting classes
  - 它们应出现在业务逻辑层。
  - 但**`Restaurant`**类依赖于**`Mapper`**类。
- Divide the Business Logic Tier into 2 Subpackages
  - 从而尽量避免了子程序包**`Persistency`**的变化对子程序包**`Domain`**带来影响。

## Persistency Architecture



## Persistency of **Table**

```
package booksys.application.domain;

public class Table {
   private int number;
   private int places;

   public Table(int n, int p) {
       number = n;
       places = p;
   }
   public int getNumber() {  return number;   }
   public int getPlaces() {  return places;   }
}
```

```
package booksys.application.persistency;
import booksys.application.domain.Table;

class PersistentTable extends Table {
   private int oid;

   PersistentTable(int oid, int tno, int places) {
      super(tno, places);
      this.oid = oid;
   }
   int getId() {
      return oid;
   }
}
```

```
package booksys.application.persistency;
import booksys.storage.Database;
import java.sql.*;
import java.util.*;

public class TableMapper {
  // Implementation of hidden cache
  private Hashtable cache;
  private void addToCache(PersistentTable t) {
    cache.put(new Integer(t.getId()), t);
  }
  private PersistentTable getFromCache(int oid) {
    return (PersistentTable) cache.get(new Integer(oid));
  }
  private PersistentTable getFromCacheByNumber(int tno) {
    PersistentTable t = null;
    Enumeration enum = cache.elements();
    while (t == null & enum.hasMoreElements()) {
      PersistentTable tmp =
        (PersistentTable) enum.nextElement();
      if (tmp.getNumber() == tno)  t = tmp;
    }
    return t;
  }
```

（转下页）

（接上页）

```
// Singleton
private TableMapper() {
    cache = new Hashtable();
}
private static TableMapper uniqueInstance;
public static TableMapper getInstance() {
    if (uniqueInstance == null) {
        uniqueInstance = new TableMapper();
    }
    return uniqueInstance;
}
```

（转下页）

（接上页）

```
// Retrievals
public PersistentTable getTable(int tno) {
    PersistentTable t = getFromCacheByNumber(tno);
    if (t == null) {
        t = getTable("SELECT * FROM 'Table' " +
            "WHERE number='" + tno + "'");
        if (t != null)  addToCache(t);
    }
    return t;
}
PersistentTable getTableForOid(int oid) {
    PersistentTable t = getFromCache(oid);
    if (t == null) {
        t = getTable("SELECT * FROM 'Table' ” +
            "WHERE oid ='" + oid + "'");
        if (t != null)  addToCache(t);
    }
    return t;
}
```

（转下页）

（接上页）

```
   private PersistentTable getTable(String sql) {
      PersistentTable t = null;
      try {
         Statement stmt = Database.getInstance().
            getConnection().createStatement();
         ResultSet rs = stmt.executeQuery(sql);
         while (rs.next())  t = new PersistentTable(
            rs.getInt(1), rs.getInt(2), rs.getInt(3)
         );
         rs.close();  stmt.close();
      } catch (SQLException e) {  e.printStackTrace();  }
      return t;
   }

   public Vector getTableNumbers() {
      Vector v = new Vector();
      try {
         Statement stmt = Database.getInstance().
            getConnection().createStatement() ;
         ResultSet rs = stmt.executeQuery(
            "SELECT * FROM 'Table' ORDER BY number");
         while (rs.next())  v.addElement(
            new Integer(rs.getInt(2))
         );
         rs.close();  stmt.close();
      } catch (SQLException e) {  e.printStackTrace();  }
      return v;
   }
}
```

## Persistency of Customer

```
package booksys.application.domain;

public class Customer {
   private String name;
   private String phoneNumber;

   public Customer(String n, String p) {
      name = n;
      phoneNumber = p;
   }
   public String getName() {
      return name;
   }
   public String getPhoneNumber() {
      return phoneNumber;
   }
}
```

```
package booksys.application.persistency;
import booksys.application.domain.Customer;

class PersistentCustomer extends Customer {
    private int oid;

    PersistentCustomer(int id, String n, String p) {
        super(n, p);
        oid = id;
    }
    int getId() {
        return oid;
    }
}
```

```
package booksys.application.persistency;
import ...

public class CustomerMapper {
    // Implementation of hidden cache
    // Singleton
    ...
    // Retrievals
    public PersistentCustomer getCustomer(
        String n, String p) ...
    PersistentCustomer getCustomerForOid(int oid) ...
    // Add a customer to the database
    PersistentCustomer createCustomer(
        String name, String phone) ...
    private PersistentCustomer getCustomer(String sql)  ...
}
```

# Using Persistent Objects

```
package booksys.application.persistency;
import ...

public class BookingMapper {
   // Singleton
   ...
   public Vector getBookings(Date date) ...
   public PersistentReservation createReservation(
      int covers, Date date, Time time,
      Table table, Customer customer, Time arrivalTime) ...
   public PersistentWalkIn createWalkIn(
      int covers,Date date,Time time,Table table) ...
   public void updateBooking(Booking b) ...
   public void deleteBooking(Booking b) ...
   private void performUpdate(String sql) ...
}
```

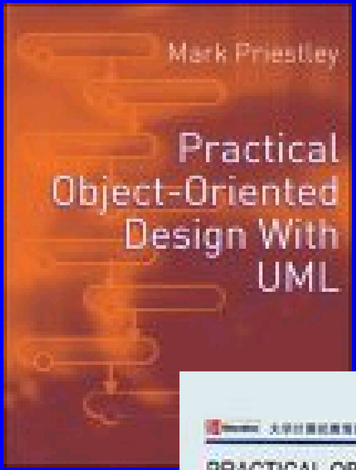
```
package booksys.application.domain;
import ...

class Restaurant {
    BookingMapper bm = BookingMapper.getInstance();
    CustomerMapper cm = CustomerMapper.getInstance();
    TableMapper tm = TableMapper.getInstance();

    Vector getBookings(Date date) {
        return bm.getBookings(date);
    }
    Customer getCustomer(String name, String phone) {
        return cm.getCustomer(name, phone);
    }
    Table getTable(int n) {
        return tm.getTable(n);
    }
    ...
}
```

# References

- These slides come from



**M. Priestley** (University of Westminster)
***Practical Object-Oriented Design with UML, 2nd Edition***
McGraw-Hill, 2004, ISBN: 0077103939

– 面向对象设计UML实践（第2版）
大学计算机教育国外著名教材系列影印版
清华大学出版社，2004年6月
ISBN：7-302-08784-9，定价39元

# ENJOY THE COURSE !