Basic Java

Unit 5 - Packages

Pratian Technologies (India) Pvt. Ltd.
www.pratian.com

Topics

- Naming Clashes
- Packages
- Creating Packages in Java
- Accessing Package Members
- Access Specifiers
- Nested Packages
- The ‘import’ statement

Naming Clashes

```
class Connection
{
        public void connectToOracle()
        {
                // Logic of connecting to
                   Oracle DB
         }
}
```

Connect to Oracle developed by team X

```
class Connection
{
        public void connectToSybase()
        {
                // Logic of connecting to
Sybase DB
        }
}
```

Connect to Sybase developed by team Y

Another Team Z uses utility classes provided by both teams X & Y

```
class DatabaseWriter
{
      public static void main(String[ ] s)
     {
         Connection connection = new
                Connection();
         connection. connectToOracle();
     }
}
```

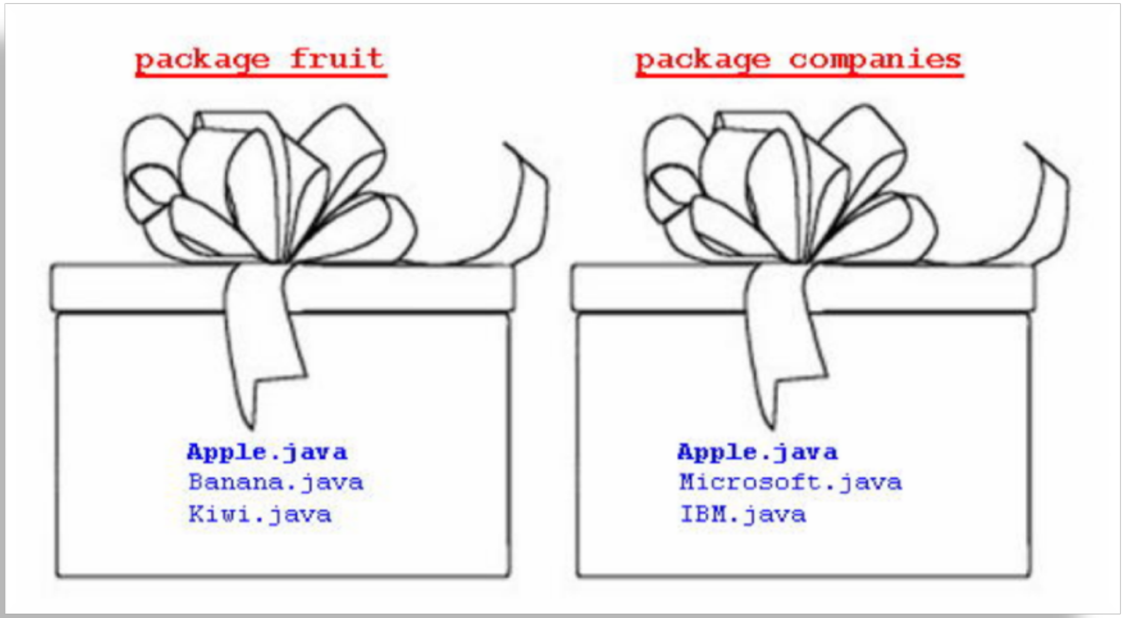
//Error

How do we solve the problem of Name clashes ?

Data[illegible]er is executed ?

Packages

- Package is a collection of related classes and interfaces.



- Packages are used to avoid naming conflicts.
- They also provide access protection.

Creating Packages

- Choose an appropriate name for the package.
- Use the *package* statement at the top of *every source file* that contains the classes or interfaces.

```
package oracleDB;
class Connection
{
        public void connectToOracle()
        { }
        ………….
}
```

```
package cybaseDB;
class Connection
{
        public void connectToSybase()
        { }
        ……………..
}
```

PRATIAN TECHNOLOGIES

Creating Packages

- Choose an appropriate name for the package.
- Use the *package* statement at the top of *every source file* that contains the classes or interfaces.

```
package oracleDB
class Connection
{
        public void connectToOracle()
        { }
        ………….
}
```

```
package cybaseDB;
class Connection
{
        public void connectToSybase()
        { }
        ……………..
}
```

Note

package statement must be the first line in the source file

Creating Packages

- Choose an appropriate name for the package.
- Use the *package* statement at the top of *every source file* that contains the classes or interfaces.

Note

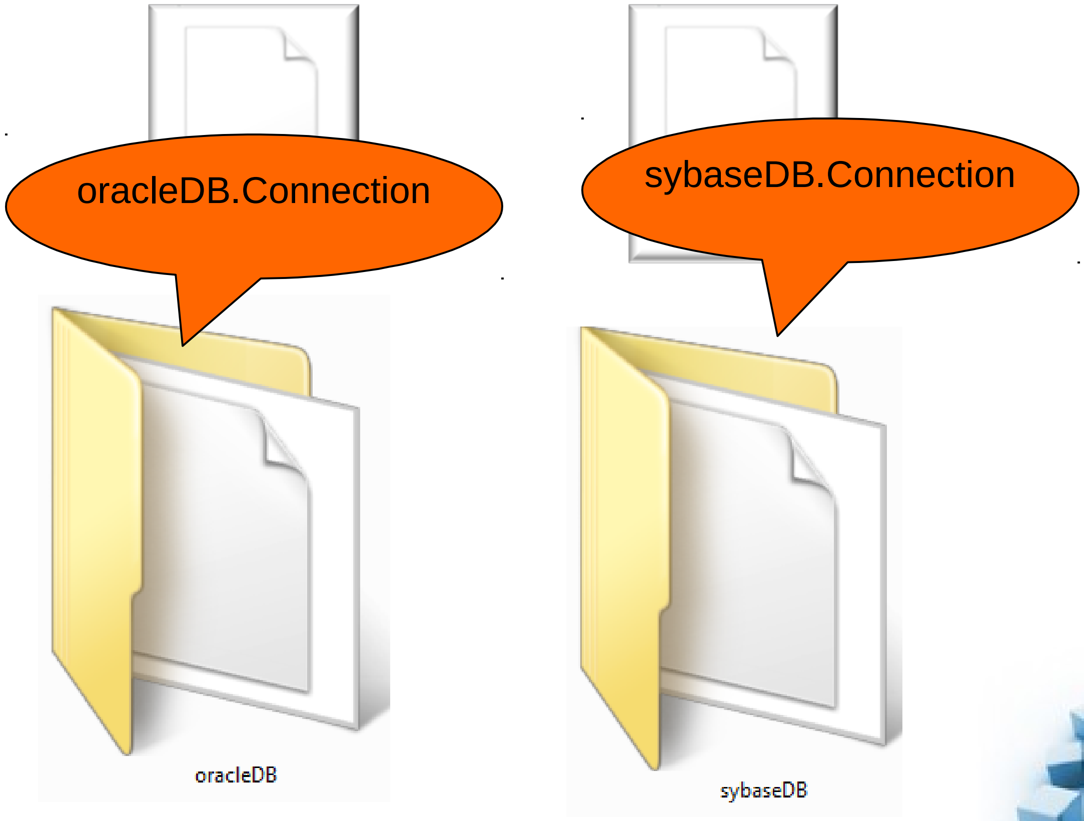
There can be only one package statement in each source file and it applies to all types in the file.

```
package oracleDB
class Connection
{
        public void connectToOracle()
        { }
        ………….
}
```

```
package cybaseDB;
class Connection
{
        public void connectToSybase()
        { }
        ……………..
}
```

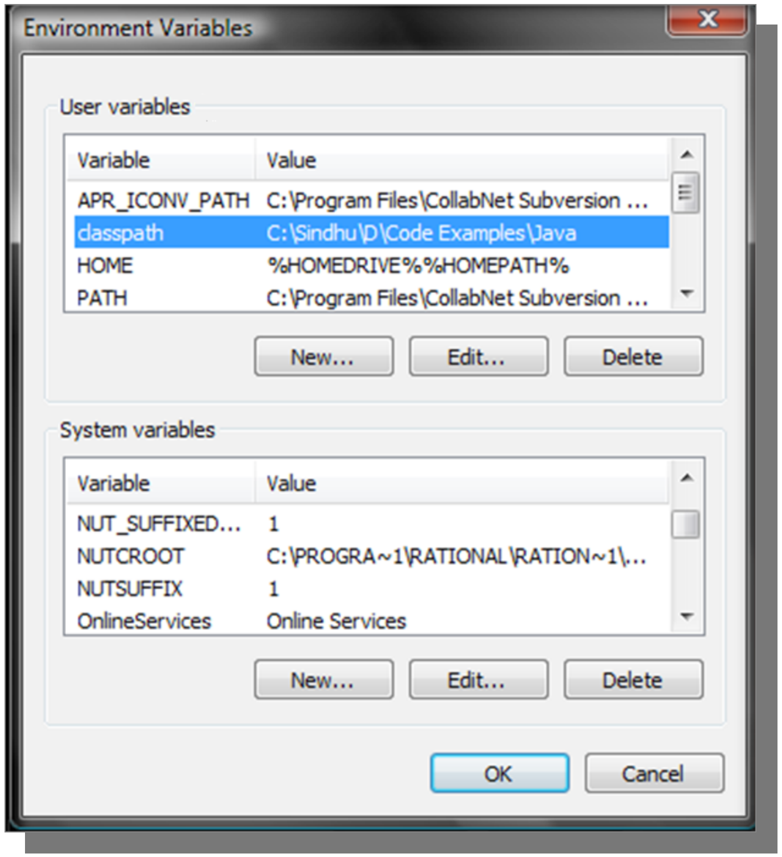
Package is a library as well !

- A physical folder with the same name as the package has to created
- All the .class files belonging to that package has to be included in that folder.



Classpath Setting

- Classpath setting is used by the JVM to locate class files
  - Class loader looks for a class file in the path mentioned and loads the bytecodes in memory
- The environment variable 'Classpath' needs to be set to the parent folder of the package folder


Environment Variables

User variables

Variable	Value
APR_ICONV_PATH	C:\Program Files\CollabNet Subversion ...
classpath	C:\Sindhu\D\Code Examples\Java
HOME	%HOMEDRIVE%%HOMEPATH%
PATH	C:\Program Files\CollabNet Subversion ...

System variables

Variable	Value
NUT_SUFFIXED...	1
NUTCROOT	C:\PROGRA~1\RATIONAL\RATION~1\...
NUTSUFFIX	1
OnlineServices	Online Services

Accessing package members

- Referring to the member by its fully qualified name

A class or interface belonging a package needs to be referred to as package name**.**class name

```
class DatabaseWriter

{

    public static void main(String[ ] s)

    {

        oracleDB.Connection conn = new
oracleDB.Connection();

        conn. connectToOracle();

    }

}
```

Creating Packages

- Summarizing steps involved in creating packages
  1. Include ‘package’ statement
  2. Create the corresponding physical folder structure
  3. Set the ‘classpath’ environment variable

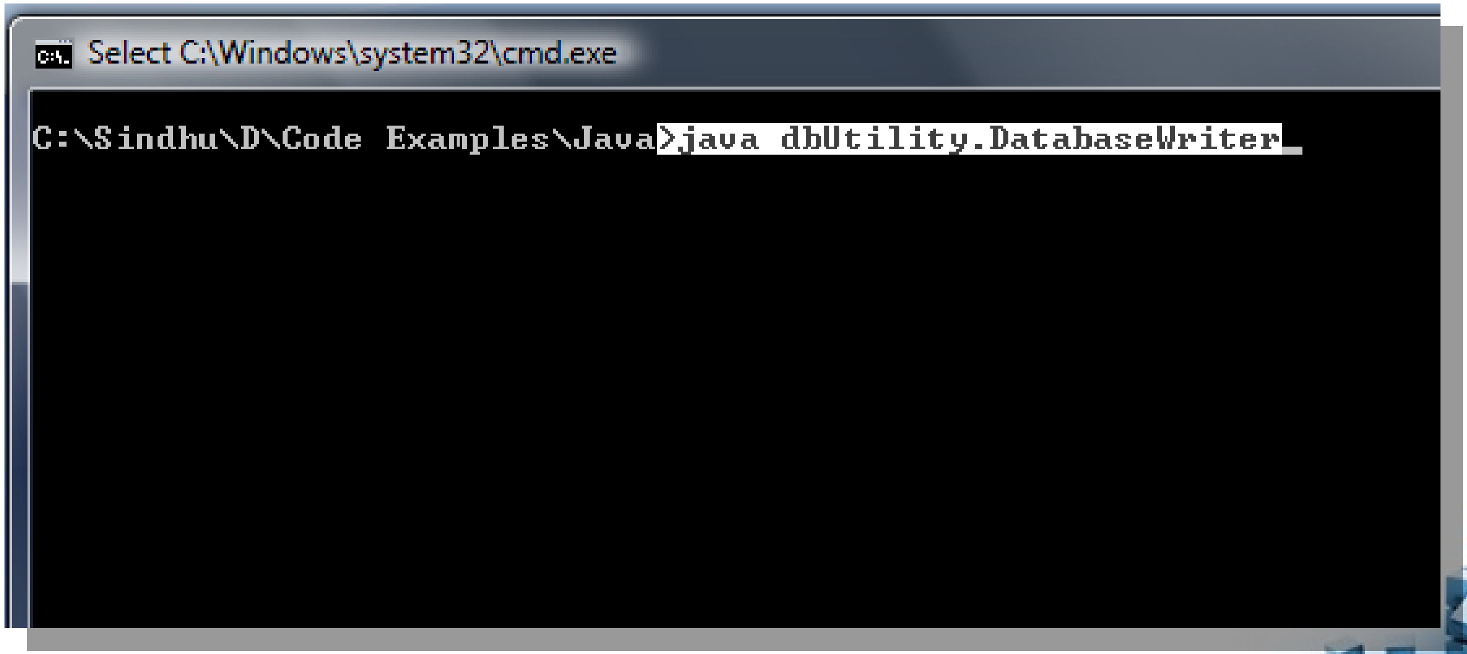
Access Specifiers - Revisited

- public
  - Any class member declared as public is visible (or accessible) to any class belonging to any package
- protected
  - Any class member declared as protected is visible (or accessible) to all classes in the same package as well as to sub classes belonging to other packages
- default
  - Any class member declared without any of the above is visible (or accessible) to all classes in the same package only.
- private
  - Any class member declared as private is visible (or accessible) only within the same class

Knowledge Check

- Include any class that you have written in a package and use the class in another class.

- Note : To run a class that belongs to a package, remember to provide the fully qualified class name as argument to the java command

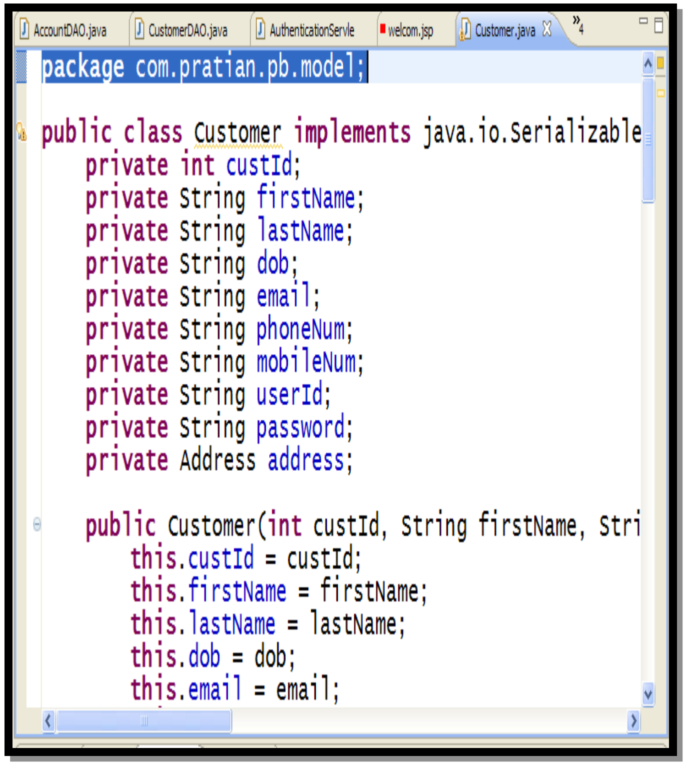


```
C:\Sindhu\D\Code Examples\Java>java dbUtility.DatabaseWriter_
```

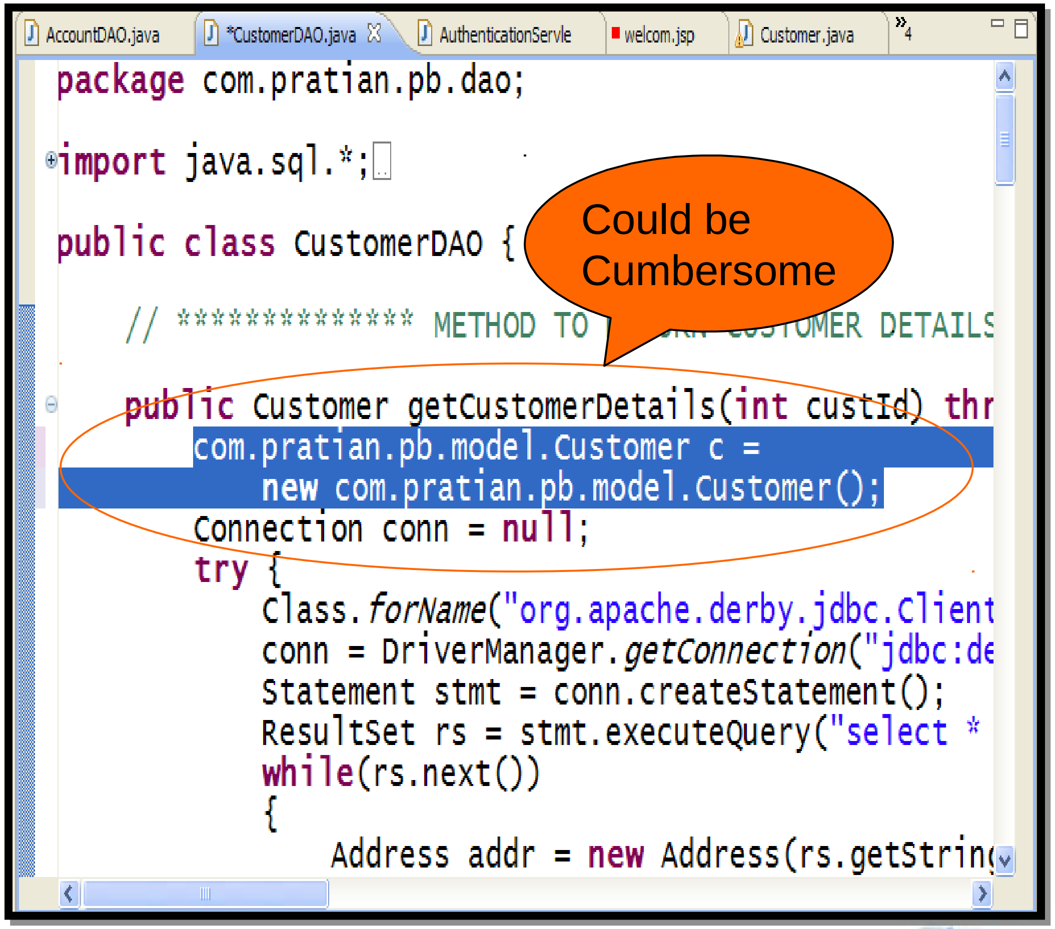
Package Naming Convention

- It is important to ensure uniqueness of package names
- Since the Internet domain names are unique for a company, the reversed domain name is used to name a package.
  - Ex com.pratian.javatraining for a package javatraining
- Package names are written in lowercase to avoid conflict with the names of classes or interfaces.
- Packages in the Java language itself begin with ***java.*** or ***javax.***

Nested Packages - Example



```
package com.pratian.pb.model;

public class Customer implements java.io.Serializable
    private int custId;
    private String firstName;
    private String lastName;
    private String dob;
    private String email;
    private String phoneNum;
    private String mobileNum;
    private String userId;
    private String password;
    private Address address;

    public Customer(int custId, String firstName, Stri
        this.custId = custId;
        this.firstName = firstName;
        this.lastName = lastName;
        this.dob = dob;
        this.email = email;
```



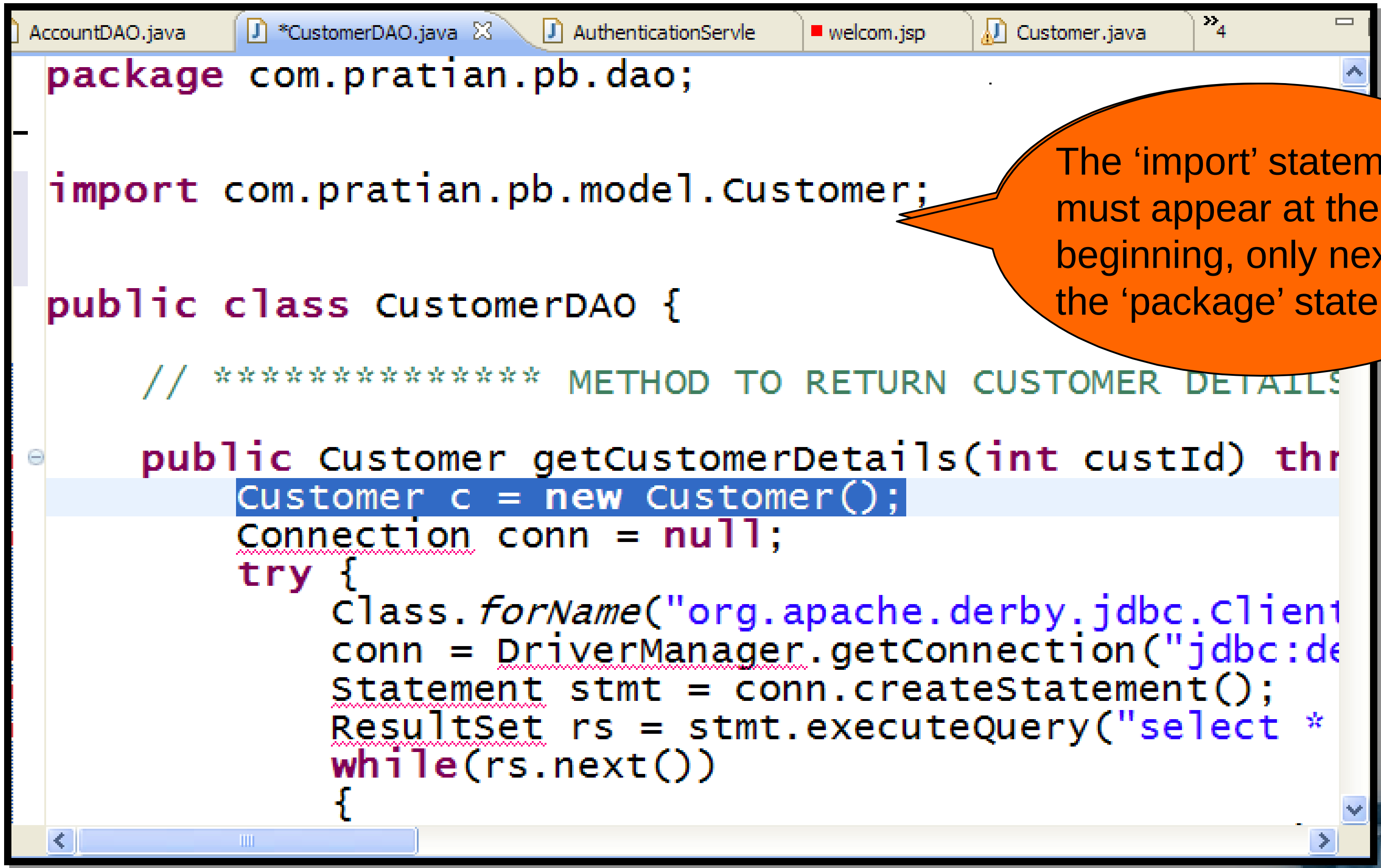
```
package com.pratian.pb.dao;

import java.sql.*;

public class CustomerDAO {

    // ************** METHOD TO RETURN CUSTOMER DETAILS

    public Customer getCustomerDetails(int custId) thr
        com.pratian.pb.model.Customer c =
            new com.pratian.pb.model.Customer();
        Connection conn = null;
        try {
            Class.forName("org.apache.derby.jdbc.Client
            conn = DriverManager.getConnection("jdbc:de
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("select *
            while(rs.next())
            {
                Address addr = new Address(rs.getStrin
```

Class Customer belongs to package **com.pratian.pb.model**

In another class CustomerDAO,it needs to be accessed as **com.pratian.pb.model.Customer**

The ‘import’ shorthand

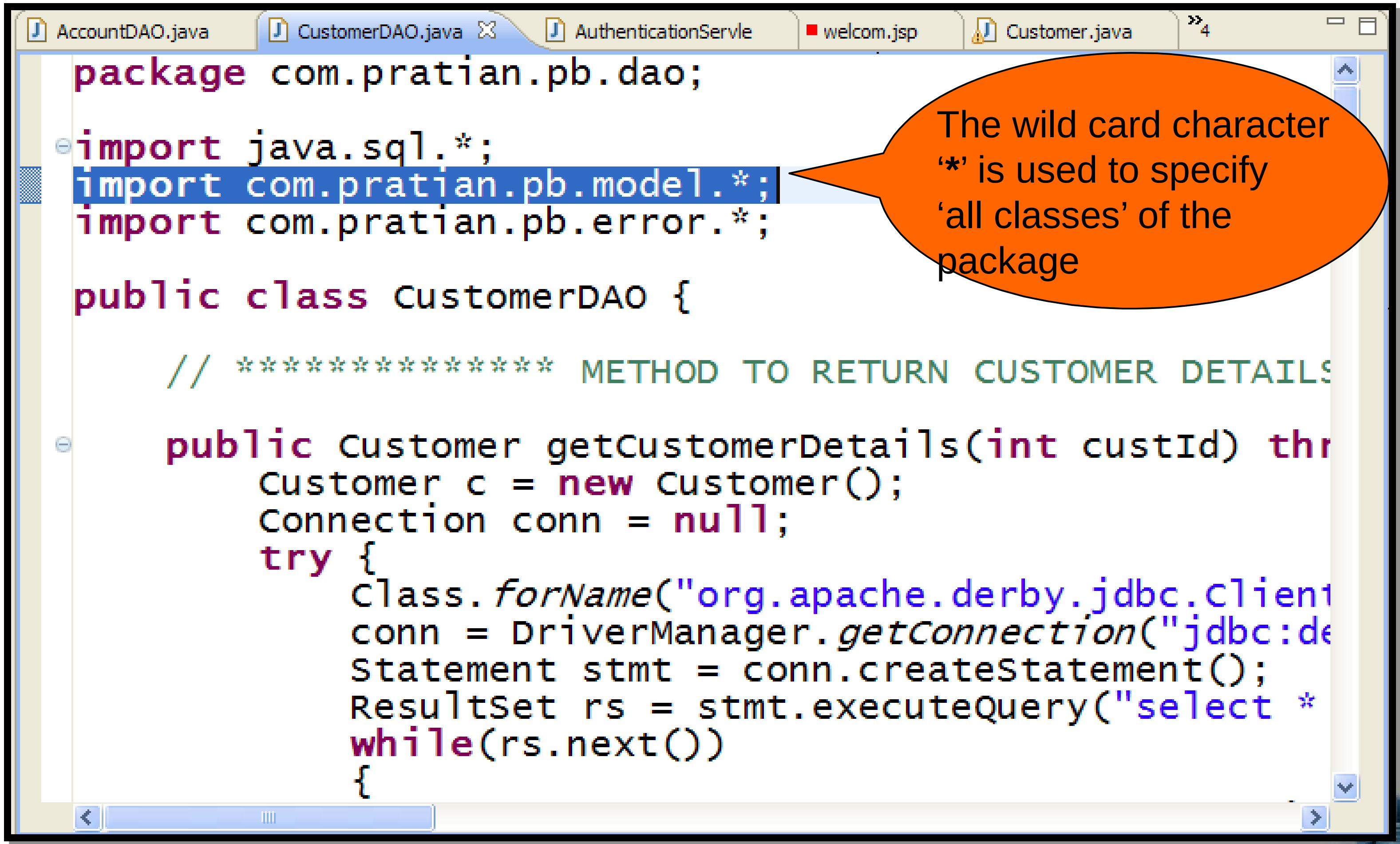
- The ‘import’ statement is a convenience syntax used to access classes that belong to a package



```
package com.pratian.pb.dao;

import com.pratian.pb.model.Customer;

public class CustomerDAO {

    // *************** METHOD TO RETURN CUSTOMER DETAILS

    public Customer getCustomerDetails(int custId) thr
        Customer c = new Customer();
        Connection conn = null;
        try {
            Class.forName("org.apache.derby.jdbc.Client
            conn = DriverManager.getConnection("jdbc:de
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("select *
            while(rs.next())
            {
```

The ‘import’ statement must appear at the beginning, only next to the ‘package’ statement

The ‘import’ shorthand



```
package com.pratian.pb.dao;

import java.sql.*;
import com.pratian.pb.model.*;
import com.pratian.pb.error.*;

public class CustomerDAO {

    // *************** METHOD TO RETURN CUSTOMER DETAILS

    public Customer getCustomerDetails(int custId) thr
        Customer c = new Customer();
        Connection conn = null;
        try {
            Class.forName("org.apache.derby.jdbc.Client
            conn = DriverManager.getConnection("jdbc:de
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery("select *
            while(rs.next())
            {
```

The wild card character ‘*’ is used to specify ‘all classes’ of the package

Nested Packages

- A package can be nested inside another package.
- A '.' separated name is used to nest package within packages.

```
package instruments.western;

public class Guitar implements
                              instruments.Instrument

{

        public void play()  {  }

}
```

- On compilation of the above code we would get a Guitar.class file, this file should be included in a directory instruments/western.

C:/myproject/instruments/western/Guitar.class

JAR Files

- The Java Archive (JAR) file format enables to bundle multiple files into a single archive file.
- Need for JAR files
  - ***Compression*** – The JAR format allows compression of files for efficient storage.
  - ***Decreased download time*** – If a library is bundled in a JAR file, the resources can be downloaded in a single transaction, without the need for opening a new connection for each file.
  - ***Portability*** - The mechanism for handling JAR files is a standard part of the Java platform's core API.

Creating a JAR File

- The basic format of the command for creating a JAR file is:

  jar cvf *jar-file input-file(s)*

- The options and arguments used in this command
  - The **c** option indicates that you want to *create* a JAR file.
  - The **f** option indicates that you want the output to go to a *file* rather than to stdout.
  - **jar-file** is the name that you want the resulting JAR file to have. By convention, JAR filenames are given a .jar extension, though this is not required.
  - The **input-file(s)** argument is a space-separated list of one or more files that you want to include in your JAR file. The input-file(s) argument can contain the wildcard * symbol. If any of the "input-files" are directories, the contents of those directories are added to the JAR archive recursively.

- This command will generate a compressed JAR file and place it in the current directory.

Viewing contents of a JAR File

- The basic format of the command for viewing the contents of a JAR file is:

```
jar tvf jar-file
```

- The options and arguments used in this command are:
  - The **t** option indicates that you want to view the *table* of contents of the JAR file.
  - The **f** option indicates that the JAR file whose contents are to be viewed is specified on the command line.
  - The **jar-file** argument is the path and name of the JAR file whose contents you want to view.
  - The **t** and **f** options can appear in either order, but there must not be any space between them.
- This command will display the JAR file's table of contents to stdout.

Adding a JAR file to class path

- While adding JAR files to the classpath, it is necessary to add, not just the location, but also the file name of the JAR file.
- E.g. – Consider a JAR file called MyLib.jar located in

  C:\MyJava\Project\MyLib.jar

  set classpath=C:\MyJava\Project\MyLib.jar

  And not

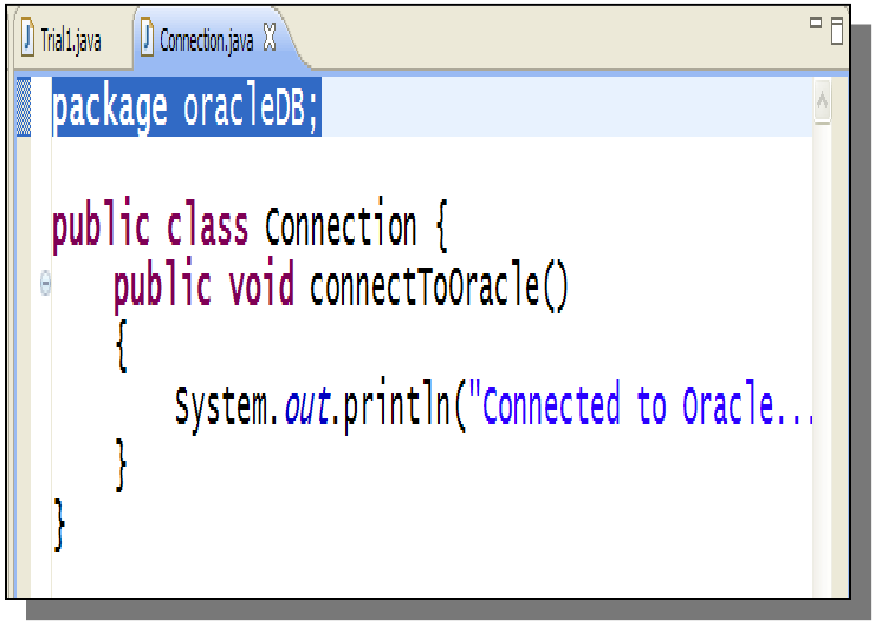
  set classpath=C:\MyJava\Project\

Question time

Please try to limit the questions to the topics discussed during the session. Thank you.

Creating Packages

- Creating packages involves the following three steps
  - i. **Include 'package' statement**

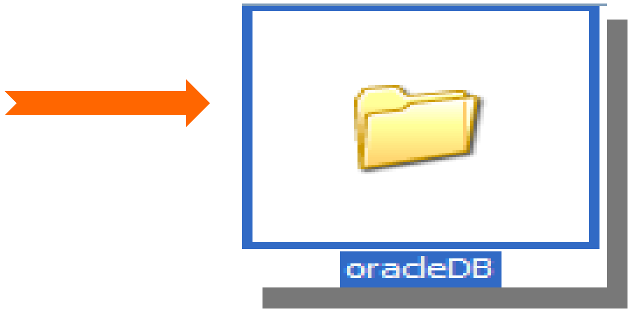


```
package oracleDB;

public class Connection {
    public void connectToOracle()
    {
        System.out.println("Connected to Oracle...
    }
}
```

  - ii. Create the corresponding physical folder structure
  - iii. Set the 'classpath' environment variable

Creating Packages

- Creating packages involves the following three steps
  - i. Include ‘package’ statement
  - ii. **Create the corresponding physical folder structure**

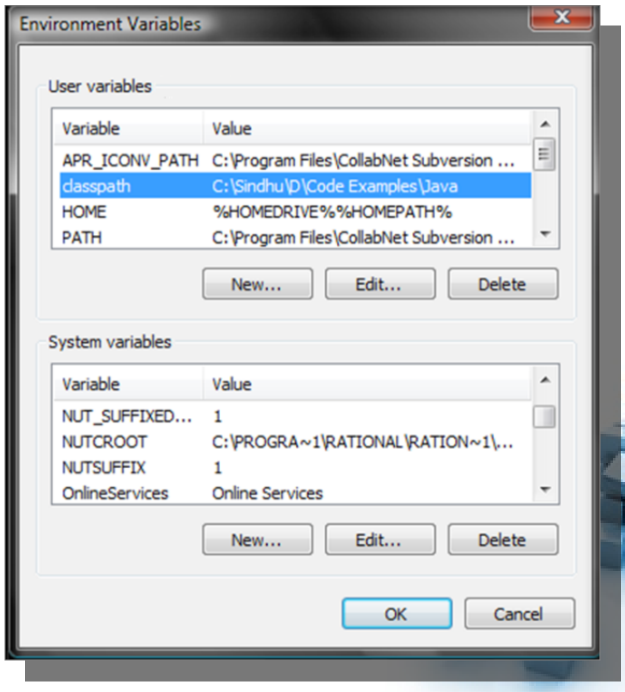


  - iii. Set the ‘classpath’ environment variable

PRATIAN TECHNOLOGIES

Creating Packages

- Creating packages involves the following three steps
  1. Include ‘package’ statement
  2. Create the corresponding physical folder structure
  3. **Set the ‘classpath’ environment variable**


Environment Variables

User variables

Variable	Value
APR_ICONV_PATH	C:\Program Files\CollabNet Subversion ...
classpath	C:\Sindhu\D\Code Examples\Java
HOME	%HOMEDRIVE%%HOMEPATH%
PATH	C:\Program Files\CollabNet Subversion ...

System variables

Variable	Value
NUT_SUFFIXED...	1
NUTCROOT	C:\PROGRA~1\RATIONAL\RATION~1\...
NUTSUFFIX	1
OnlineServices	Online Services