

# 15

## Deploying and Maintaining the Soccer Application

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

# Objectives

After completing this lesson, you should be able to:

- Deploy a simple application as a JAR file
- Describe the parts of a Java application, including the user interface and the back end
- Describe how classes can be extended to implement new capabilities in the application



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

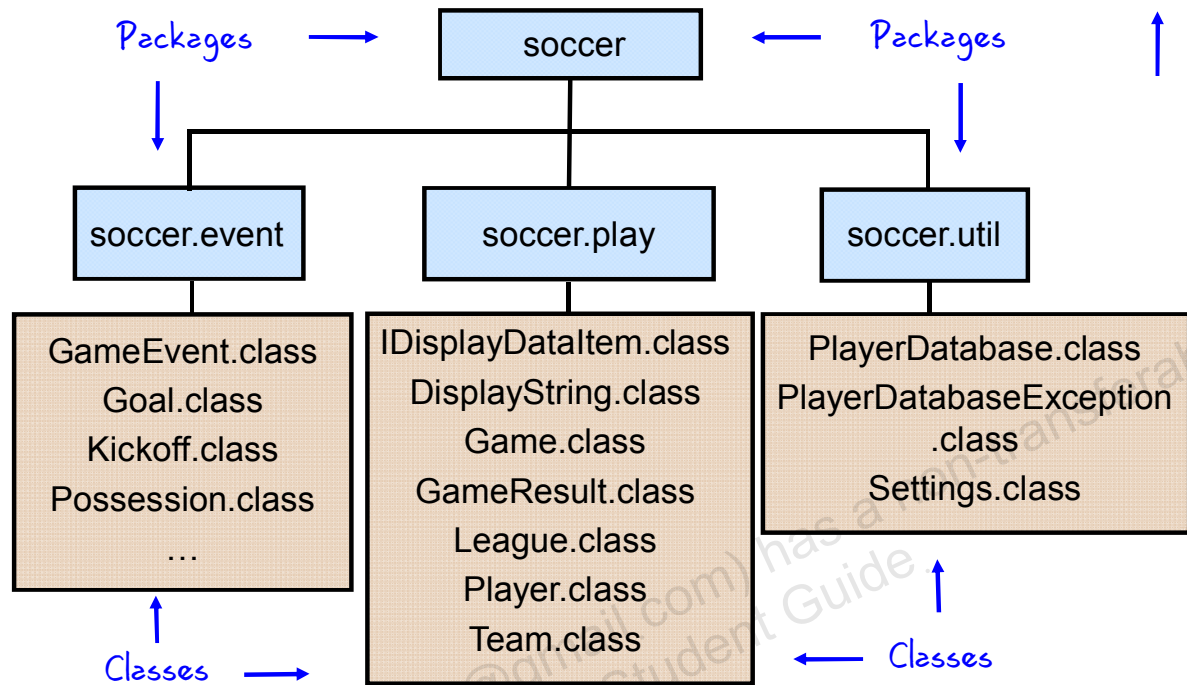
# Topics

- Packages
- JARs and deployment
- Two-tier and three-tier architecture
- The Soccer application
- Application modifications and enhancements

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

# Packages



ORACLE

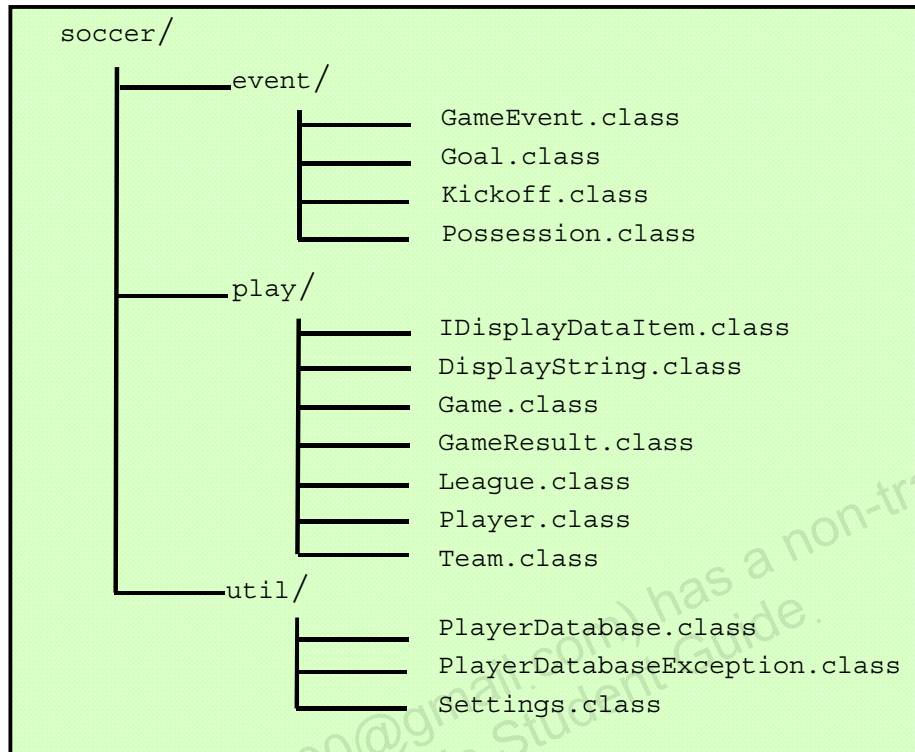
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Classes are grouped into packages to ease management of the system.

There are many ways to group classes into meaningful packages. There is no right or wrong way, but a common technique is to group classes into a package by semantic similarity.

For example, the software for the soccer application could contain a set of event classes (the superclass `GameEvent`, with subclasses `Goal`, `Kickoff`, and so on), a set of classes that use these event classes to model the playing of a game, and a set of utility classes. All these packages are contained in the top-level package called `soccer`.

# Packages Directory Structure

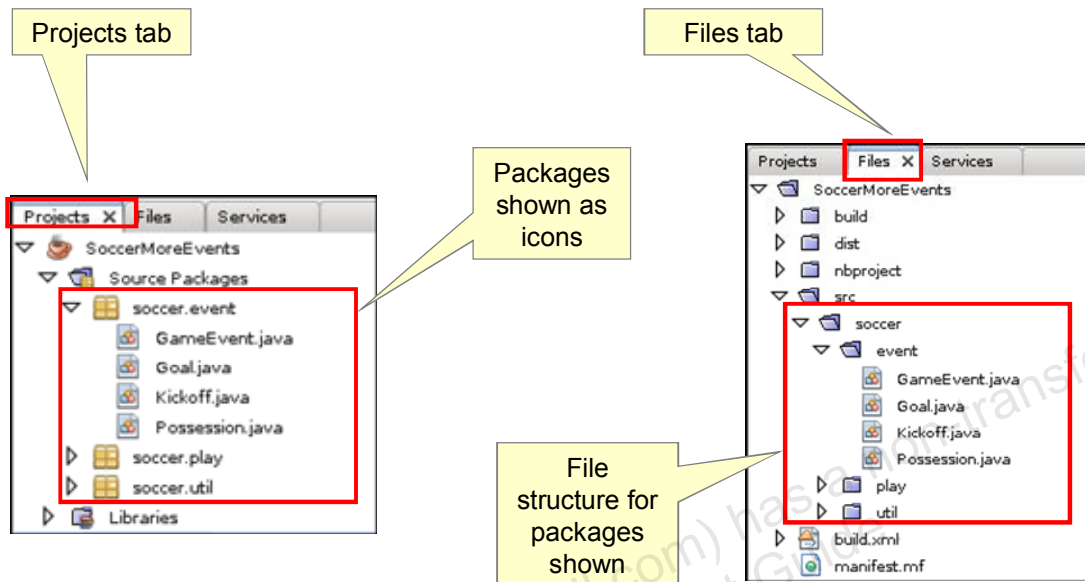


ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Packages are stored in a directory tree containing directories that match the package names. For example, the `Goal.class` file should exist in the directory `event`, which is contained in the directory `soccer`.

# Packages in NetBeans



ORACLE

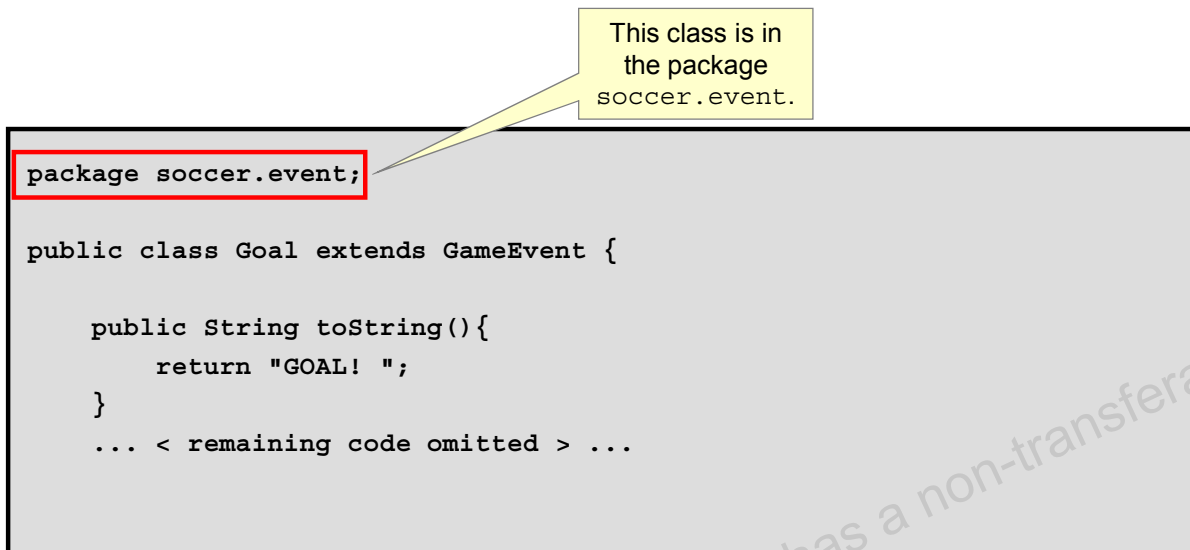
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The left panel in NetBeans has three tabs. Two of these tabs, Projects and Files, show how packages relate to the file structure.

The Projects tab shows the packages and libraries for each project. The source package shown is the one containing the packages and classes for the Soccer application, and the screenshot shows the three packages: `soccer.event`, `soccer.play`, and `soccer.util`. Each of these packages can be expanded to show the source files within, as has been done for the `soccer.event` package in the screenshot.

The Files tab shows the directory structure for each project. In the screenshot, you can see how the packages listed on the Projects tab have a corresponding directory structure. For example, the `soccer.events` package has the corresponding file structure of the `duke` directory just under the `src` directory and contains the `item` directory, which in turn contains all the source files in the package.

# Packages in Source Code



```
package soccer.event;  
  
public class Goal extends GameEvent {  
  
    public String toString(){  
        return "GOAL! ";  
    }  
    ... < remaining code omitted > ...  
}
```

This class is in the package soccer.event.

The package that a class belongs to is defined in the source code.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The example code in the slide shows the package statement being used to define the package that the `Goal` class is in. Just as the class itself must be in a file of the same name as the class, the file (in this case, `Goal.java`) must be contained in a directory structure that matches the package name.

# Topics

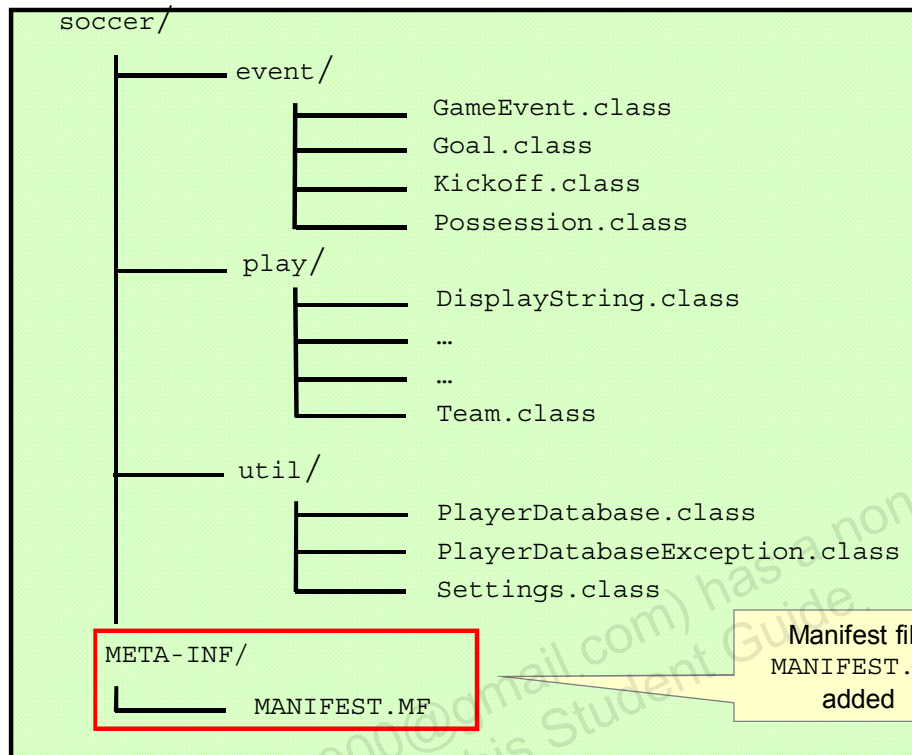
- Packages
- JARs and deployment
- Two-tier and three-tier architecture
- The Soccer application
- Application modifications and enhancements

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.



# SoccerEnhanced.jar



The JAR file contains the entire class directory including the manifest file.

ORACLE

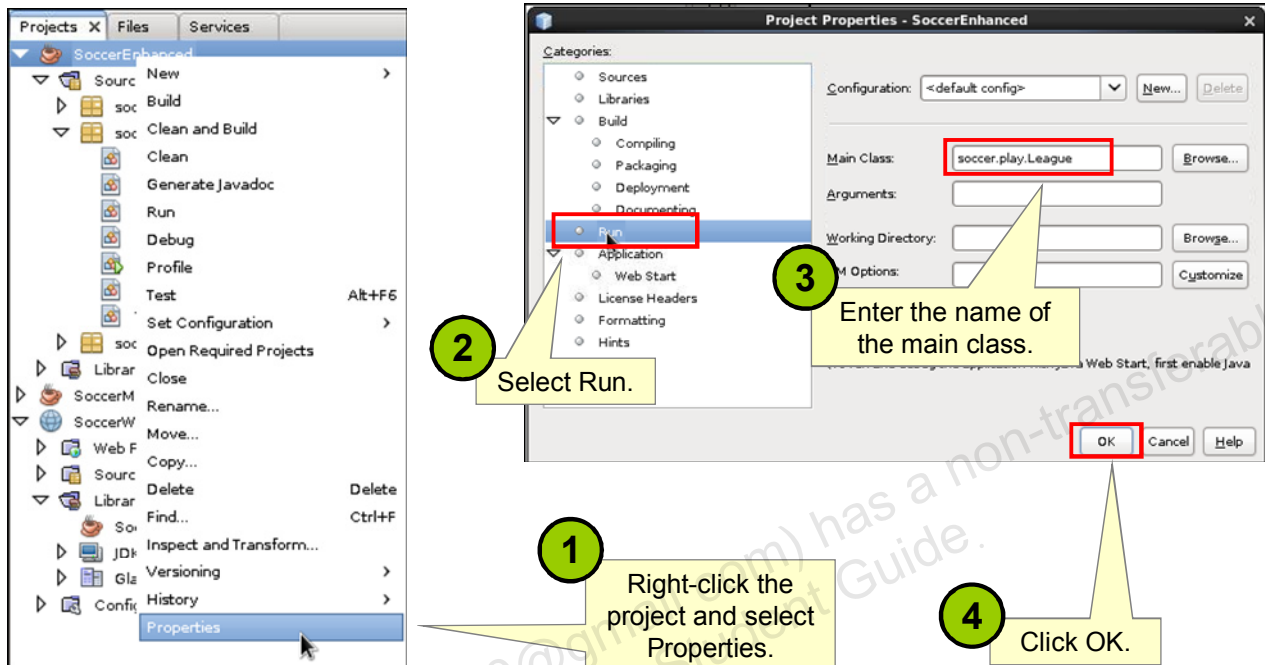
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To deploy a Java application, you typically put the necessary files into a JAR file. This greatly simplifies running the application on another machine.

A JAR file is much like a zip file (or a tar file on UNIX) and contains the entire directory structure for the compiled classes plus an additional `MANIFEST.MF` file in the `META-INF` directory. This `MANIFEST.MF` file tells the Java runtime which file contains the `main` method.

You can create a JAR file by using a command-line tool called `jar`, but most IDEs make the creation easier. In the following slides, you see how to create a JAR file using NetBeans.

## Set Main Class of Project

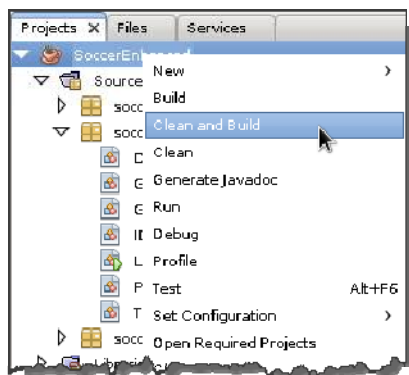


ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Before you create the JAR file, you need to indicate which file contains the `main` method. This is subsequently written to the `MANIFEST.MF` file.

# Creating the JAR File with NetBeans

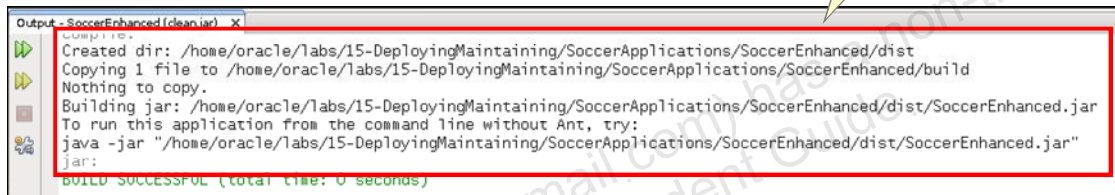


1

Right-click the project and select "Clean and Build."

2

Check the output to ensure that the build is successful.



ORACLE

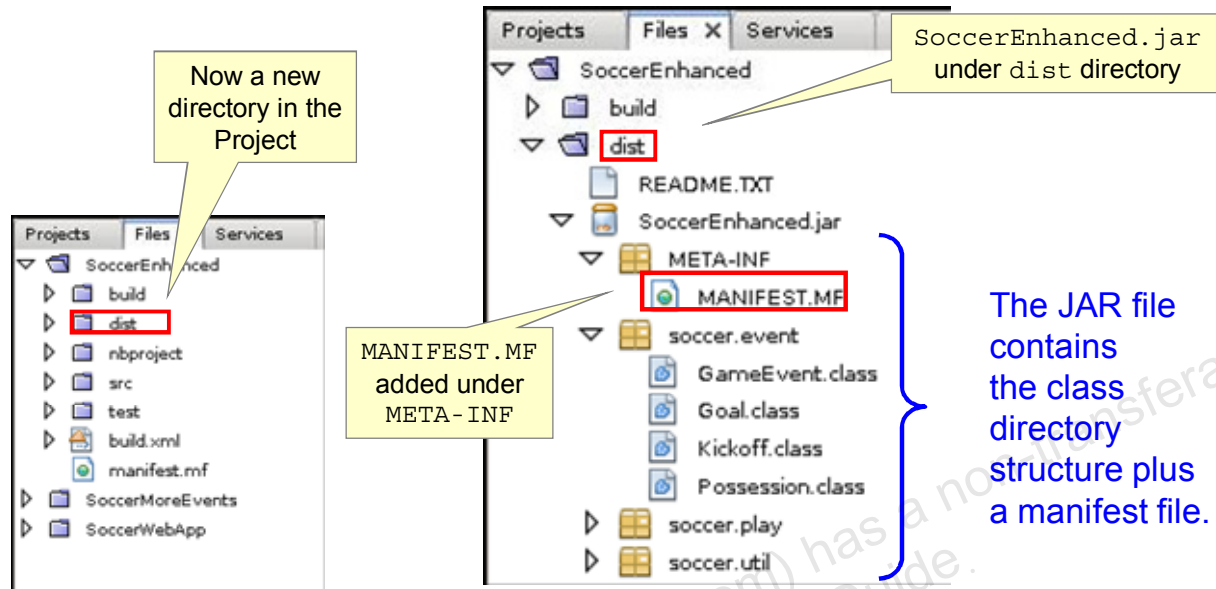
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

You create the JAR file by right-clicking the project and selecting "Clean and Build." For a small project such as SoccerEnhanced, this should take only a few seconds.

- Clean removes any previous builds.
- Build creates a new JAR file.

You can also run "Clean" and "Build" separately.

## Creating the JAR File with NetBeans



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

By default, the JAR file will be placed in the dist directory. (This directory is removed in the clean process and re-created during build.) Using the Files tab of NetBeans, you can look inside the JAR file and make sure that all the correct classes have been added.

# Topics

- Packages
- JARs and deployment
- **Two-tier and three-tier architecture**
- The Soccer application
- Application modifications and enhancements

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

# Client/Server Two-Tier Architecture

Client/server computing involves two or more computers sharing tasks:

- Each computer performs logic appropriate to its design and stated function.
- The front-end client communicates with the back-end database.
- The client requests data from the back end.
- The server returns the appropriate results.
- The client handles and displays data.

ORACLE

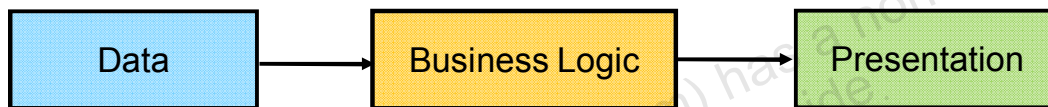
Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

A major performance penalty is paid in two-tier client/server. The client software ends up larger and more complex because most of the logic is handled there. The use of server-side logic is limited to database operations. The client here is referred to as a *thick client*.

Thick clients tend to produce frequent network traffic for remote database access. This works well for intranet-based and local area network (LAN)–based network topologies, but produces a large footprint on the desktop in terms of disk and memory requirements. Also, not all back-end database servers are the same in terms of server logic offered, and all of them have their own API sets that programmers must use to optimize and scale performance.

# Client/Server Three-Tier Architecture

- Three-tier client/server is a more complex, flexible approach.
- Each tier can be replaced by a different implementation:
  - The data tier is an encapsulation of all existing data sources.
  - Business logic defines business rules.
  - Presentation can be GUI, web, smartphone, or even console.



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

The three components or tiers of a three-tier client/server environment are *data*, *business logic* or *functionality*, and *presentation*. They are separated so that the software for any one of the tiers can be replaced by a different implementation without affecting the other tiers.

For example, if you want to replace a character-oriented screen (or screens) with a GUI (the presentation tier), you write the GUI using an established API or interface to access the same functionality programs in the character-oriented screens.

The business logic offers functionality in terms of defining all of the business rules through which the data can be manipulated. Changes to business policies can affect this layer without having an impact on the actual databases.

The third tier, or data tier, includes existing systems, applications, and data that have been encapsulated to take advantage of this architecture with minimal transitional programming effort.

# Topics

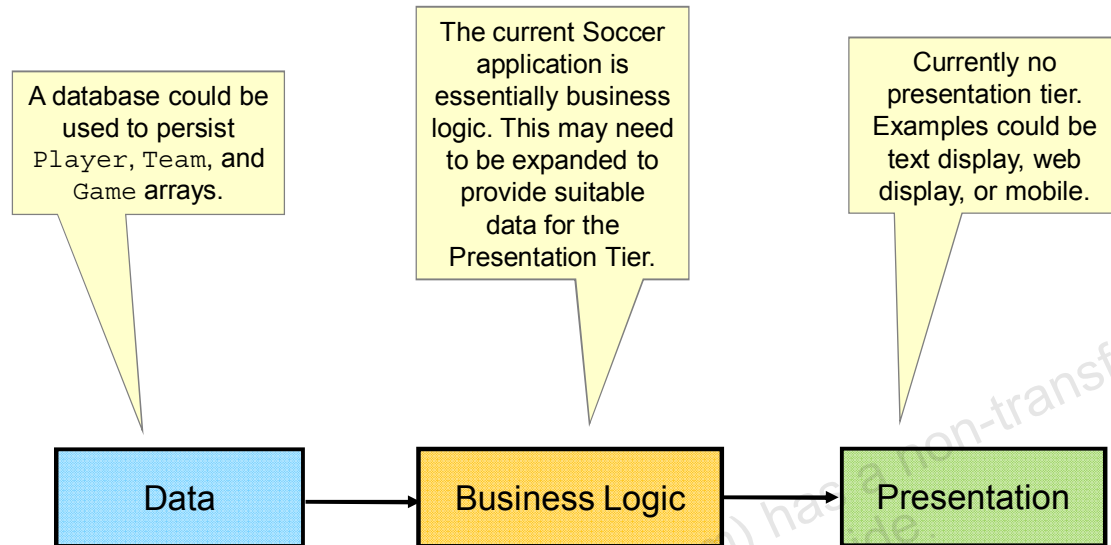
- Packages
- JARs and deployment
- Two-tier and three-tier architecture
- **The Soccer application**
- Application modifications and enhancements

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.



# Client/Server Three-Tier Architecture

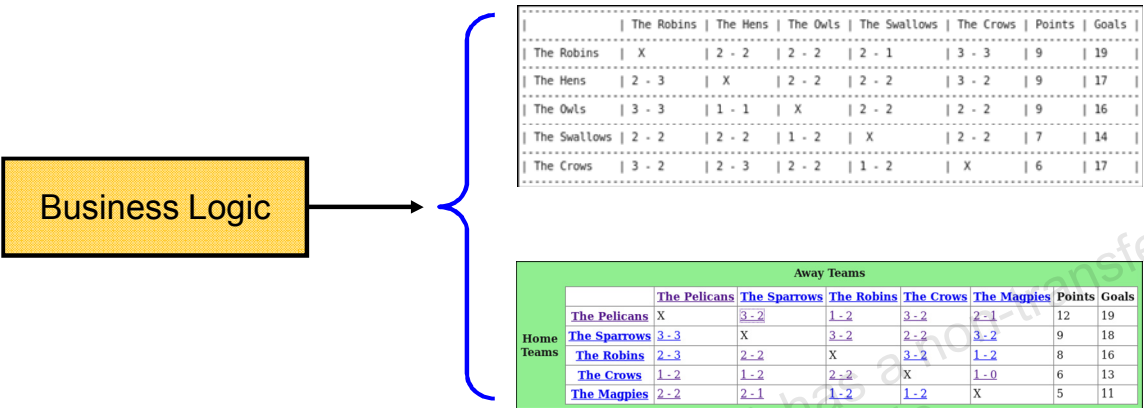


ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the Soccer application, the Data tier does not really exist—there is currently no way to persist the data. But a Data tier could be added by saving the `Player`, `Team`, and `Game` arrays to a database. The current code of the Soccer application is business logic code. There is currently no presentation tier—at the moment the only presentation of data is the console of the application. To support a presentation tier, the business logic tier may need to present the data in some fashion where it can be consumed easily by many different types of presentation tier.

# Client/Server Three-Tier Architecture



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

To support different presentations of the grid view of the league, the business logic tier should provide the data in a way that it can be queried to produce the individual values needed.

## Different Outputs

A two-dimensional `String` array could provide the `String` output for each element of the grid, but this is inflexible:

- The presentation can only display the `String` provided.
- The presentation cannot access other useful information—for example, the data required to allow users to click on the score for more details.

	The Robins	The Hens	The Owls	The Swallows	The Crows	Points	Goals
The Robins	X	2 - 2	2 - 2	2 - 1	3 - 3	9	19
The Hens	2 - 3	X	2 - 2	2 - 2	3 - 2	9	17
The Owls	3 - 3	1 - 1	X	2 - 2	2 - 2	9	16
The Swallows	2 - 2	2 - 2	1 - 2	X	2 - 2	7	14
The Crows	3 - 2	2 - 3	2 - 2	1 - 2	X	6	17

Away Teams		The Pelicans	The Sparrows	The Robins	The Crows	The Magpies	Points	Goals
Home Teams	The Pelicans	X	3 - 2	1 - 2	3 - 2	2 - 1	12	19
	The Sparrows	3 - 3	X	3 - 2	2 - 2	3 - 2	9	18
	The Robins	2 - 3	2 - 2	X	3 - 2	1 - 2	8	16
	The Crows	1 - 2	1 - 2	2 - 2	X	1 - 0	6	13
	The Magpies	2 - 2	2 - 1	1 - 2	1 - 2	X	5	11

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Ideally for each element of data in the two-dimensional array, the presentation should have access to the data the display is based on, or at least a useful subset of that data. For example, for the text that displays a team name, the presentation might wish to query further data about the team—for example to provide a pop-up list of the players in the team or a pop-up of the details of a game for any of the game scores.

Note that using a two-dimensional array is of course not the only way to package the data; you could use a `List` of `List` objects or create a custom class.

But assuming a two-dimensional array—it can only be of one type, so you cannot put references to `Team` objects and a `Game` objects into the same array. Or can you?

# The Soccer Application

- Abstract classes
  - `GameEvent`
    - Extended by `Goal` and other `GameEvent` classes
- Interfaces
  - `Comparable`
    - Implemented by `Team` and `Player` so that they can be ranked
  - `IDisplayDataItem`
    - Implemented by `Team`, `Game`, and `DisplayString`

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

An enhanced version of the Soccer application has been created to illustrate object-oriented programming in Java. The `IDisplayDataItem` is a new Interface that is implemented by `Team` and `Game`, and a new class, `DisplayString`. Any class that implements this interface can be used in a two-dimensional array of type `IDisplayDataItem`. So you can have references that access both `Team` objects and `Game` objects in the same array after all!

## IDisplayDataItem Interface

```
package soccer.play;

public interface IDisplayDataItem {

    public boolean isDetailAvailable ();
    public String getDisplayDetail();
    public int getID();
    public String getDetailType();

}
```

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

# Running the JAR File from the Command Line

```

Output - SoccerEnhanced (clean.jar) X
Created dir: /home/oracle/labs/15-DeployingMaintaining/SoccerApplications/SoccerEnhanced/dist
Copying 1 file to /home/oracle/labs/15-DeployingMaintaining/SoccerApplications/SoccerEnhanced/dist
Nothing to copy.
Building jar: /home/oracle/labs/15-DeployingMaintaining/SoccerApplications/SoccerEnhanced/dist/SoccerEnhanced.jar
To run this application from the command line without Ant, try:
java -jar "/home/oracle/labs/15-DeployingMaintaining/SoccerApplications/SoccerEnhanced/dist/SoccerEnhanced.jar"
BUILD SUCCESSFUL (total time: 0 seconds)

```

```
[oracle@EDBSR2P14 ~]$ java -jar "/home/oracle/labs/15-DeployingMaintaining/SoccerApplications/SoccerEnhanced/dist/SoccerEnhanced.jar"
```

	The Robins	The Hens	The Owls	The Swallows	The Crows	Points	Goals
The Robins	X	2 - 2	2 - 2	2 - 1	3 - 3	9	19
The Hens	2 - 3	X	2 - 2	2 - 2	3 - 2	9	17
The Owls	3 - 3	1 - 1	X	2 - 2	2 - 2	9	16
The Swallows	2 - 2	2 - 2	1 - 2	X	2 - 2	7	14
The Crows	3 - 2	2 - 3	2 - 2	1 - 2	X	6	17

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Running the command-line application using the JAR file is very straightforward and the instructions are actually given in the output window for the build process. (If the JAR were a GUI application, it would be run the same way.)

If the application were an early command-line version of the software, you might run it as shown in the slide. Even though the output is simply to a terminal, the display code is iterating through a two-dimensional array of type `IDisplayDataItem` to build this display.

# Text Presentation of the League

```
[oracle@EDBSR2P14 ~]$ java -jar "/home/oracle/labs/15-DeployingMaintaining/SoccerApplications/SoccerEnhanced/dist/SoccerEnhanced.jar"
```

	The Robins	The Hens	The Owls	The Swallows	The Crows	Points	Goals
The Robins	X	2 - 2	2 - 2	2 - 1	3 - 3	9	19
The Hens	2 - 3	X	2 - 2	2 - 2	3 - 2	9	17
The Owls	3 - 3	1 - 1	X	2 - 2	2 - 2	9	16
The Swallows	2 - 2	2 - 2	1 - 2	X	2 - 2	7	14
The Crows	3 - 2	2 - 3	2 - 2	1 - 2	X	6	17

The object type behind these data elements is Team.

The object type behind these data elements (except for the output Xs) is Game.

The object type behind these data elements is DisplayString.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Note that there is a new class, `DisplayString`, that also implements `IDisplayDataItem`. It is used for data that is not represented by any of the current core classes.

# Web Presentation of the League

		Away Teams						Home Teams
		The Pelicans	The Sparrows	The Robins	The Crows	The Magpies	Points	Goals
The Pelicans	X		3 - 2	1 - 2	3 - 2	2 - 1	12	19
The Sparrows	3 - 3	X		3 - 2	2 - 2	3 - 2	9	18
The Robins	2 - 3	2 - 2	X		3 - 2	1 - 2	8	16
The Crows	1 - 2	1 - 2	2 - 2	X		1 - 0	6	13
The Magpies	2 - 2	2 - 1	1 - 2	1 - 2	X		5	11

The object type behind these data elements is Team.

The object type behind these data elements (except for the output Xs) is Game.

The object type behind these data elements is DisplayString.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

Given that there is a two-dimensional array of type `IDisplayDataItem`, it can just as easily be used to create the output for a web display.



# Topics

- Packages
- JARs and deployment
- Two-tier and three-tier architecture
- The Soccer application
- Application modifications and enhancements

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

## Enhancing the Application

- Well-designed Java software minimizes the time required for:
  - Maintenance
  - Enhancements
  - Upgrades
- For the Soccer application, it should be easy to:
  - Add new `GameEvent` subclasses (business logic)
  - Develop new clients (presentation)
    - Take the application to a smartphone (for example)
  - Change the storage system (data)

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

In the following slides, you see what is involved in adding another class to represent a new `GameEvent`.

## Adding a New `GameEvent` Kickoff

It is possible to add a new `GameEvent` to record kickoffs by:

- Creating a new `Kickoff` class that extends the `GameEvent` class
- Adding any new unique features for the item
- Modifying any other classes that need to know about this new class

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

## Game Record Including Kickoff

The Magpies vs. The Sparrows (2 - 3)			
Event	Team	Player	Time
Kickoff	The Sparrows	Dorothy Parker	0
Possession	The Sparrows	Jane Austin	15
Possession	The Sparrows	J. M. Synge	19
Possession	The Sparrows	Brendan Behan	20
Possession	The Sparrows	Dorothy Parker	26
GOAL!	The Sparrows	Dorothy Parker	32
Kickoff	The Magpies	G. K. Chesterton	34
Possession	The Magpies	Oscar Wilde	35
Possession	The Magpies	G. K. Chesterton	41
GOAL!	The Magpies	G. K. Chesterton	43
Kickoff	The Sparrows	Dorothy Parker	50
Possession	The Sparrows	J. M. Synge	54
GOAL!	The Sparrows	J. M. Synge	55
Kickoff	The Magpies	Wilkie Collins	59
Possession	The Magpies	G. K. Chesterton	62
Possession	The Magpies	Arthur Conan Doyle	63
Possession	The Magpies	Oscar Wilde	64
GOAL!	The Magpies	Oscar Wilde	74
Kickoff	The Sparrows	Frank O'Connor	75
Possession	The Sparrows	Frank O'Connor	81
GOAL!	The Sparrows	Frank O'Connor	83

The new event, Kickoff, has been added.

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

# Summary

In this lesson, you should have learned how to:

- Deploy a simple application as a JAR file
- Describe the parts of a Java application, including the user interface and the back end
- Describe how classes can be extended to implement new capabilities in the application



ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

# Course Summary

In this course, you should have learned how to:

- List and describe several key features of the Java technology: object-oriented, multithreaded, distributed, simple, and secure
- Identify different Java technology groups
- Describe examples of how Java is used in applications as well as in consumer products
- Describe the benefits of using an integrated development environment (IDE)
- Develop classes and describe how to declare a class
- Analyze a business problem to recognize objects and operations that form the building blocks of the Java program design

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

# Course Summary

- Define the term *object* and its relationship to a class
- Demonstrate Java programming syntax
- Write a simple Java program that compiles and runs successfully
- Declare and initialize variables
- List several primitive data types
- Instantiate an object and effectively use object reference variables
- Use operators, loops, and decision constructs
- Declare and instantiate arrays and ArrayLists and be able to iterate through them

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.

# Course Summary

- Use Javadocs to look up Java foundation classes
- Declare a method with arguments and return values
- Use inheritance to declare and define a subclass of an existing superclass
- Describe how errors are handled in a Java program
- Describe how to deploy a simple Java application by using the NetBeans IDE

ORACLE

Copyright © 2014, Oracle and/or its affiliates. All rights reserved.