Learning Java - A Foundational Journey

Session: 6

Modifiers and Packages

Objectives

- Describe field and method modifiers
- Explain different types of modifiers
- Explain rules and best practices for using field modifiers
- Describe class variables
- Explain creation of static variables and methods
- Describe package and its advantages
- Explain creation of user-defined package
- Explain creation of .jar files for deployment

Introduction

- Java is a tightly encapsulated language.
- Java provides a set of access specifiers such as `public`, `private`, `protected`, and `default` that help to restrict access to class and class members.
- Java provides additional field and method modifiers to further restrict access to the members of a class to prevent modification by unauthorized code.
- Java provides the concept of class variables and methods to create a common data member that can be shared by all objects of a class as well as other classes.
- Java provides packages that can be used to group related classes that share common attributes and behavior.
- The entire set of packages can be combined into a single file called the `.jar` file for deployment on the target system.

Field and Method Modifiers

Field and method modifiers are keywords used to identify fields and methods that provide controlled access to users.

Some of these can be used in conjunction with access specifiers such as `public` and `protected`.

- Different field modifiers that can be used are as follows:
  - `volatile`
  - `native`
  - `transient`
  - `final`

‘volatile’ Modifier 1-2

The `volatile` modifier allows the content of a variable to be synchronized across all running threads.

A thread is an independent path of execution of code within a program. Many threads can run concurrently within a program.

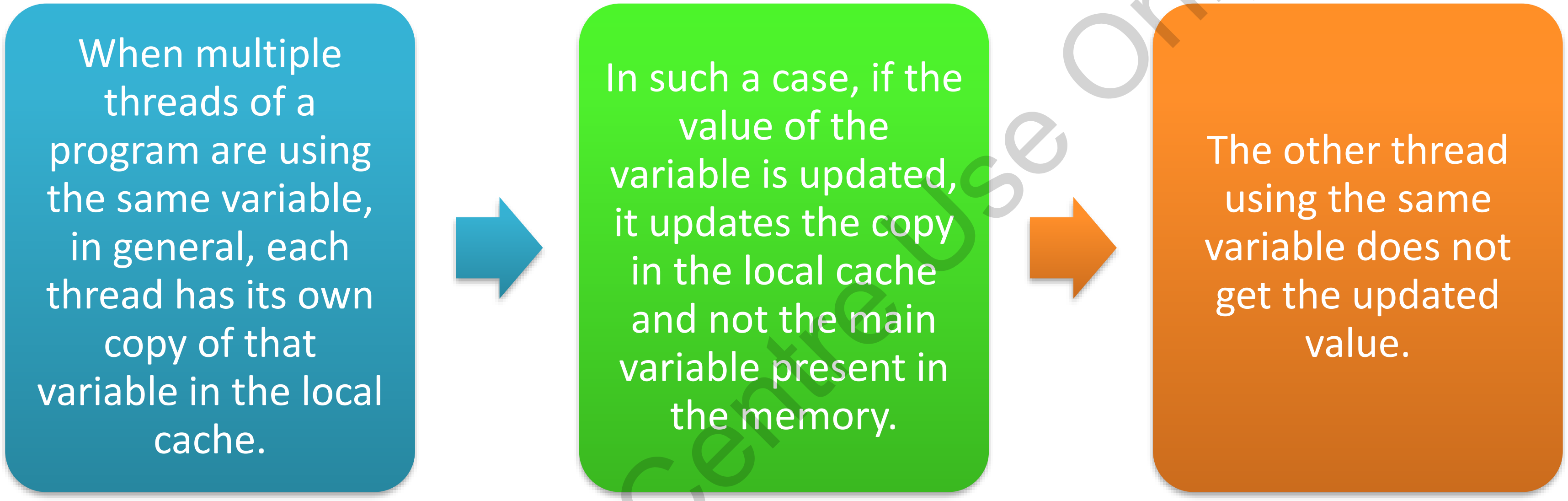
The `volatile` modifier is applied only to fields.

Constructors, methods, classes, and interfaces cannot use this modifier.

The `volatile` modifier is not frequently used.

While working with a multithreaded program, the `volatile` keyword is used.

‘volatile’ Modifier 2-2



When multiple threads of a program are using the same variable, in general, each thread has its own copy of that variable in the local cache.

In such a case, if the value of the variable is updated, it updates the copy in the local cache and not the main variable present in the memory.

The other thread using the same variable does not get the updated value.

- Whenever a thread updates the values of the variable, it updates the variable present in the main memory.
- For example,

```
private volatile int testValue; // volatile variable
```

‘native’ Modifier 1-2

- The `native` modifier is used only with methods.
- Constructors, fields, classes, and interfaces cannot use this modifier.
- The methods declared using the `native` modifier are called native methods.
- The implementation of the method exists in a library outside the JVM.

- Before invoking a native method, the library that contains method implementation must be loaded by making following system call:
  `System.loadLibrary(“libraryName”);`

Code Snippet:

```
class NativeModifier {
native void nativeMethod(); // declaration of a native method
/**
* static code block to load the library
*/
static {
System.loadLibrary("NativeMethodDefinition");
}
/**
* @param args the command line arguments
*/
public static void main(String[] args) {
NativeModifier objNative = new NativeModifier(); // line1
objNative.nativeMethod(); // line2
}
}
```

'native' Modifier 2-2

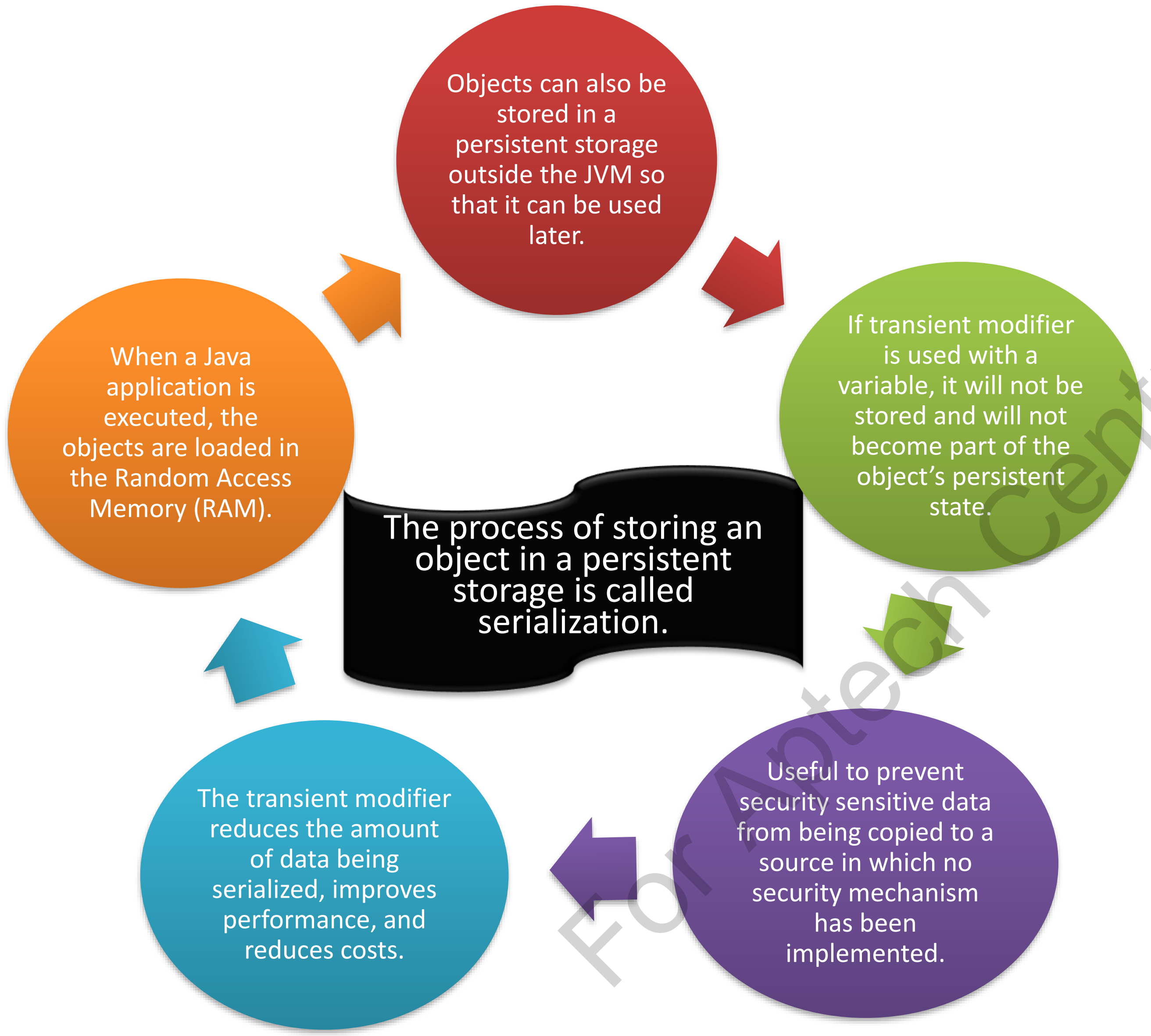
Impending security risk

- The native method executes actual machine code and therefore, it can gain access to any part of the host system.
- That is, the native code is not restricted to the JVM execution environment.

Loss of portability

- The native code is bundled in a DLL, so that it can be loaded on the machine on which the Java program is executing.
- Each native method is dependent on the CPU and the OS.

‘transient’ Modifier


Objects can also be stored in a persistent storage outside the JVM so that it can be used later.
If transient modifier is used with a variable, it will not be stored and will not become part of the object’s persistent state.
Useful to prevent security sensitive data from being copied to a source in which no security mechanism has been implemented.
The transient modifier reduces the amount of data being serialized, improves performance, and reduces costs.
When a Java application is executed, the objects are loaded in the Random Access Memory (RAM).
The process of storing an object in a persistent storage is called serialization.

Code Snippet:

```
class Circle {
transient float PI; // transient
// variable that will not persist
float area; // instance variable that
// will persist
. . .
. . .
}
```

'final' Modifier 1-3

The `final` modifier is used when modification of a class or data member is to be restricted.

The `final` modifier can be used with a variable, method, and class.

Final Variable

A `final` variable is assigned a value at the time of declaration.

- Following code snippet shows the creation of a `final` variable:

```
final float PI = 3.14;
```

'final' Modifier 2-3

Final Method

A `final` method is commonly used to generate a random constant in a mathematical application.

- Following code snippet depicts the creation of a `final` method:

```
final float getCommission(float sales){
  System.out.println("A final method. . .");
}
```

Final Class

A class declared `final` cannot be inherited or subclassed.

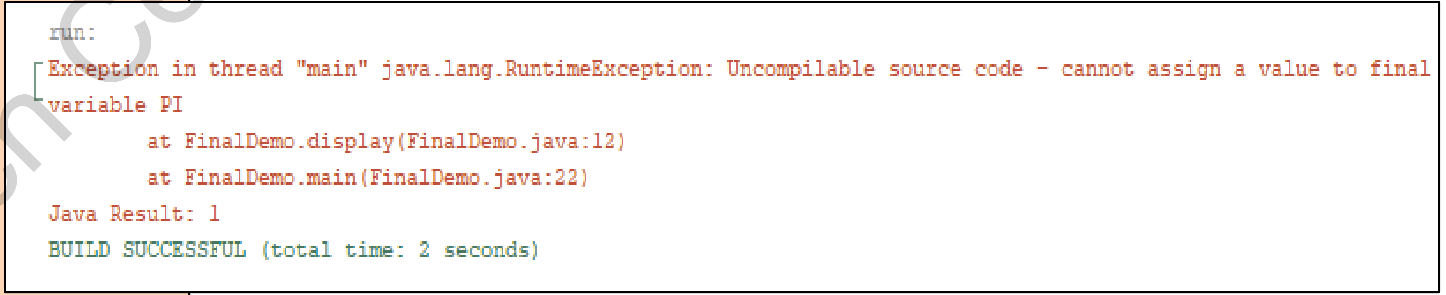
- Following code snippet shows the creation of a `final` class:

```
public final class Stock {
...
}
```

‘final’ Modifier 3-3

```
public class FinalDemo {
// Declare and initialize a final variable
final float PI = 3.14F; // variable to store value of PI
/**
* Displays the value of PI
*
* @param pi a float variable storing the value of PI
* @return void
*/
public void display(float pi) {
PI = pi; // generates compilation error
System.out.println("The value of PI is:"+PI);
}
/**
* @param args the command line arguments
*/
public static void main(String[] args) {
// Instantiate the FinalDemo class
final FinalDemo objFinalDemo = new FinalDemo();
// Invoke the display() method
objFinalDemo.display(22.7F);
}
}
```

- The class **FinalDemo** consists of a final float variable **PI** set to **3.14**. The method **display()** is used to set a new value passed by the user to PI. However, this leads to compilation error ‘**cannot assign a value to final variable PI**’.
- If the user chooses to run the program anyway, the following runtime error is issued as shown in figure:



```
run:
Exception in thread "main" java.lang.RuntimeException: Uncompilable source code - cannot assign a value to final
variable PI
        at FinalDemo.display(FinalDemo.java:12)
        at FinalDemo.main(FinalDemo.java:22)
Java Result: 1
BUILD SUCCESSFUL (total time: 2 seconds)
```

Rules and Best Practices for Using Field Modifiers

- Some of the rules for using field modifiers are as follows:

`Final` fields cannot be `volatile`**.**

`Native` **methods in Java cannot have a body.**

Declaring a `transient` **field as** `static` **or** `final` **should be avoided as far as possible.**

`Native` **methods violate Java's platform independence characteristic.**

A `transient` **variable may not be declared as** `final` **or** `static`**.**

Class Variables

Consider a situation, where in a user wants to create a counter that keeps track of the number of objects accessing a particular method.

In such a scenario, a variable is required that can be shared among all the objects of a class and any changes made to the variable are updated only in one common copy.

Java provides the implementation of such a concept of class variables by using the `static` keyword.

Declaring Class Variables

Class variables are also known as `static` variables.

Such variables are associated with the class rather than any object.

All instances of the class share the same value of the class variable.

A `static` variable declared as `final` becomes a constant whose value cannot be modified.

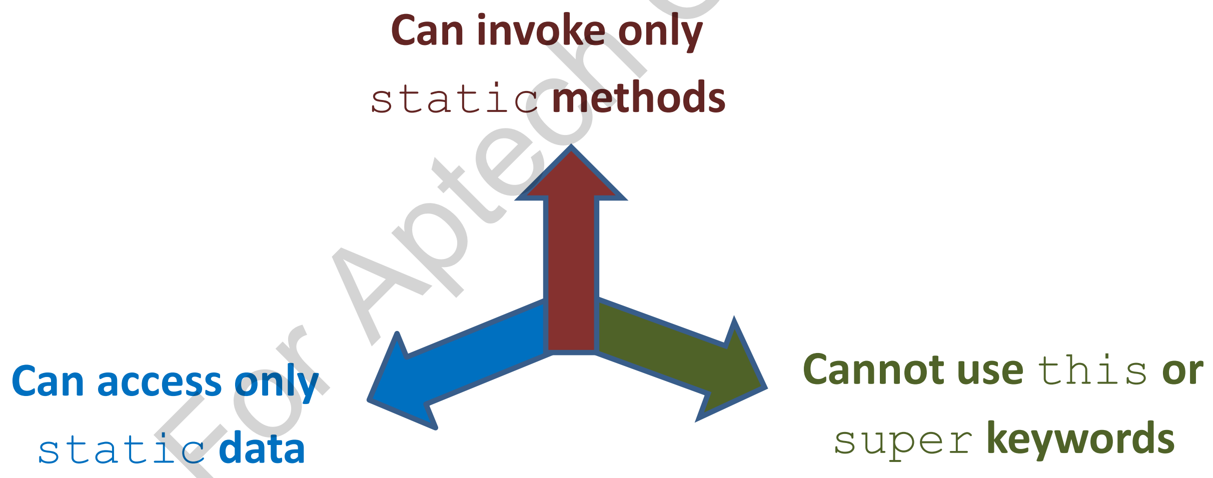
Creating Static Variables, Static Methods, and Static Blocks 1-3

One can also create `static` methods and `static` initializer blocks along with `static` variables.

`Static` variables and methods can be manipulated without creating an instance of the class.

A `static` method can only access `static` variables and not instance variables.

- Methods declared as `static` have the following restrictions:


Can invoke only static methods
Can access only static data
Cannot use this or super keywords

Creating Static Variables, Static Methods, and Static Blocks 2-3

Generally, a constructor is used to initialize variables.

It is used when a block of code needs to be executed during loading of the class by JVM.

Execution of Java code starts from `static` blocks and not from `main()` method. It is enclosed within {} braces.

There can be more than one `static` block in a program. They can be placed anywhere in the class.

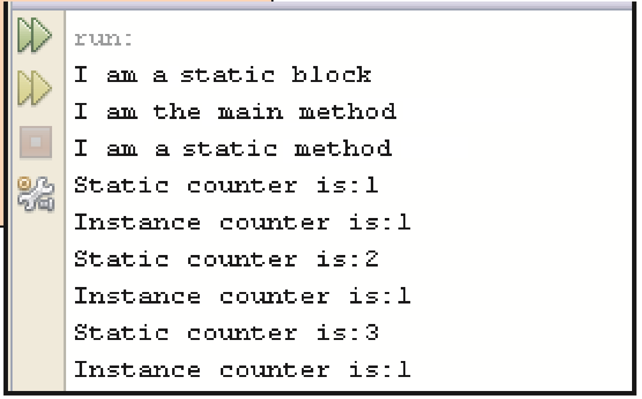
A `static` initialization block can reference only those class variables that have been declared before it.

Creating Static Variables, Static Methods, and Static Blocks 3-3

```
package session6;
public class StaticMembers {
// Declare and initialize static variable
public static int staticCounter = 0;
// Declare and initialize instance variable
int instanceCounter = 0;
/**
* static block
*
*/
static{
System.out.println("I am a static block");
}
/**
* Static method
*
* @return void
*/
public static void staticMethod(){
System.out.println("I am a static method");
}
/**
* Displays the value of static and instance counters
*
* @return void
*/
public void displayCount(){
//Increment the static and instance variable
staticCounter++;
```

```
instanceCounter++;
// Print the value of static and instance variable
System.out.println("Static counter is:"+ staticCounter);
System.out.println("Instance counter is:"+ instanceCounter);
}
/**
* @param args the command line arguments
*/
public static void main(String[] args) {
System.out.println("I am the main method");
// Invoke the static method using the class name
StaticMembers.staticMethod();
// Create first instance of the class
StaticMembers objStatic1 = new StaticMembers();
objStatic1.displayCount();
// Create second instance of the class
StaticMembers objStatic2 = new StaticMembers();
objStatic2.displayCount();
// Create third instance of the class
StaticMembers objStatic3 = new StaticMembers();
objStatic3.displayCount();
}
}
```

Output:



```
run:
I am a static block
I am the main method
I am a static method
Static counter is:1
Instance counter is:1
Static counter is:2
Instance counter is:1
Static counter is:3
Instance counter is:1
```

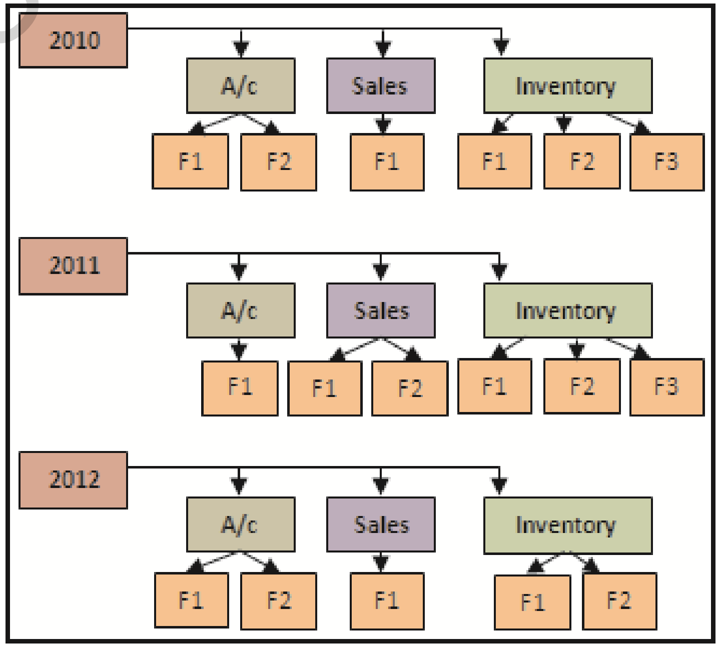
Packages 1-2

Packages

Consider a situation where in a user has about fifty files of which some are related to sales, others are related to accounts, and some are related to inventory.

Also, the files belong to different years. All these files are kept in one section of a cupboard.

Now, when a particular file is required, the user has to search the entire cupboard. This is very time consuming and difficult.

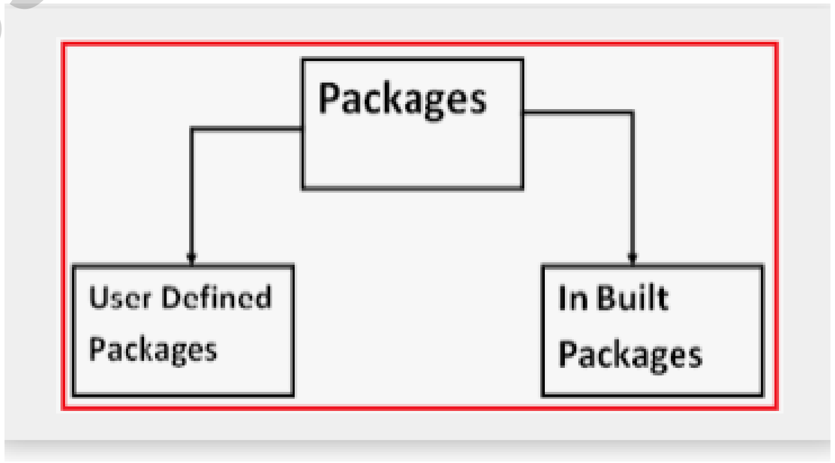


Packages 2-2

A package is a namespace that groups related classes and interfaces and organizes them as a unit.

A package can have sub packages.

A package cannot have two members with the same name.



Package names are written in lowercase.

Standard packages in the Java language begin with java or javax.

Advantages of Using Packages

One can easily determine that these classes are related.

One can know where to find the required type that can provide the required functions.

One can allow classes within one package to have unrestricted access to one another while restricting access to classes outside the package.

Packages can also store hidden classes that can be used within the package, but are not visible or accessible outside the package.

When a program from a package is called for the first time, the entire package gets loaded into the memory.

Type of Packages 1-4

The `String` class stores the state and behavior related to character strings.

The `File` class allows the developer to create, delete, compare, inspect, or modify a file on the file system.

The `Socket` class allows the developer to create and use network sockets.

Various Graphical User Interface (GUI) control classes such as `Button`, `Checkbox`, and so on provide ready to use GUI controls.

- Different types of Java packages are as follows:

Predefined packages **User-defined packages**

- Predefined packages that are part of Java API are:

`java.io` `java.util` `java.awt`

Type of Packages 2-4

- To create user-defined packages, perform following steps:

1 • Select an appropriate name for the package.

2 • Create a folder with the same name as the package.

3 • Place the source files in the folder created for the package.

4 • Add the `package` statement as the first line in all source files under that package as depicted in following code snippet:

```
package session6;
class StaticMembers{
 public static void main(String[] args)
 {}
}
```

5 • Save the source file **`StaticMembers.java`** in the package **`session6`**.

6 • Compile the code as follows:

`javac StaticMembers.java` OR Compile the code with `–d` option as follows:

```
javac -d . StaticMembers.java
```

7 • From the parent folder of the source file, execute it using the fully qualified name as follows:

```
java session6.StaticMembers
```

Type of Packages 3-4

- Java allows the user to import the classes from predefined as well as user-defined packages using an import statement.

- A member of a `public` class can be accessed outside the package by doing any of the following:

  - Referring to the member class by its fully qualified name, that is,
    `package-name.class-name.`
  - Importing the package member, that is,
    `import package-name.class-name.`
  - Importing the entire package, that is,
    `import package-name.*.`

Type of Packages 4-4

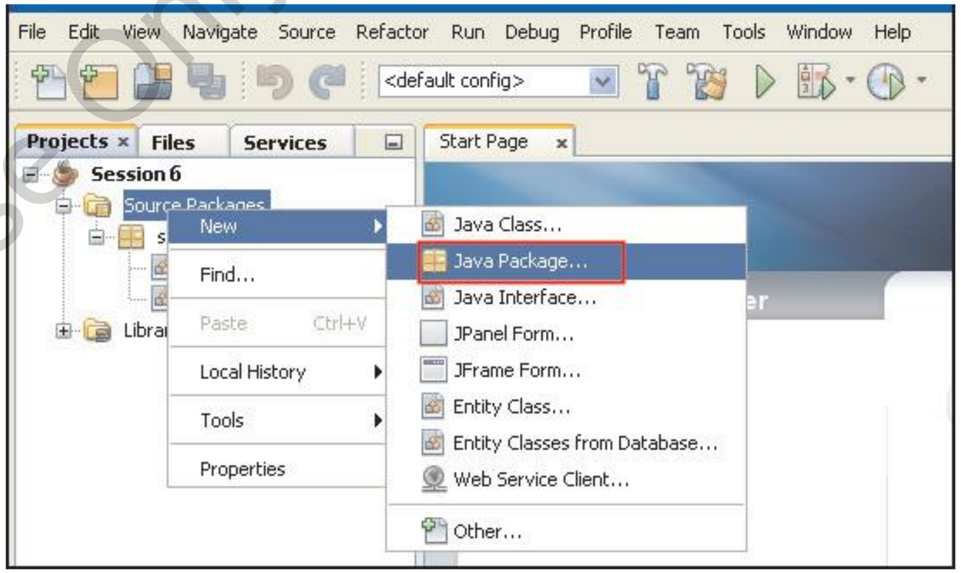
To create a new package using NetBeans IDE, perform following steps:

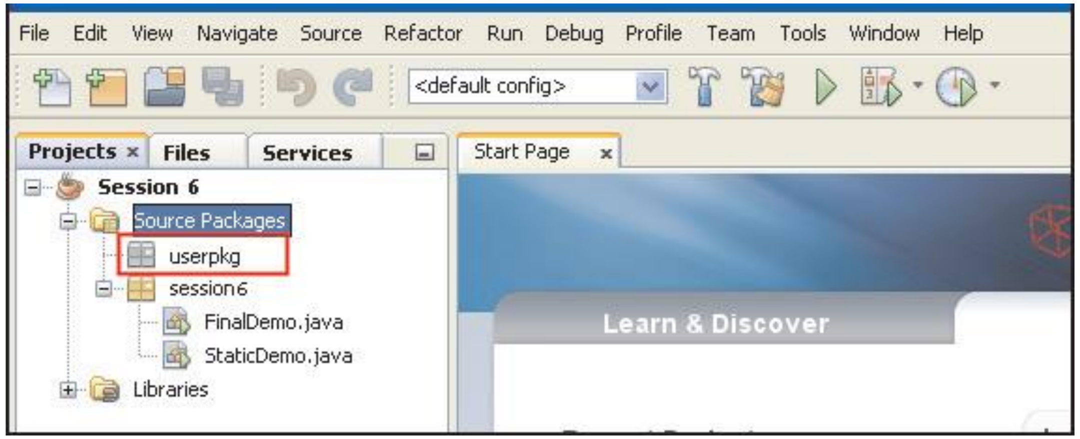
1. Open the project in which the package is to be created. In this case, `Session6` project has been chosen.

2. Right-click **Source Packages** → **New** → **Java Package** to display the **New Java Package** dialog box. For example, in figure, the project **Session6** is opened and the **Java Package** option is selected.

3. Type **userpkg** in the **Package Name** box of the **New Java Package** dialog box that is displayed.

4. Click **Finish**. The **userpkg** package is created as shown in figure.



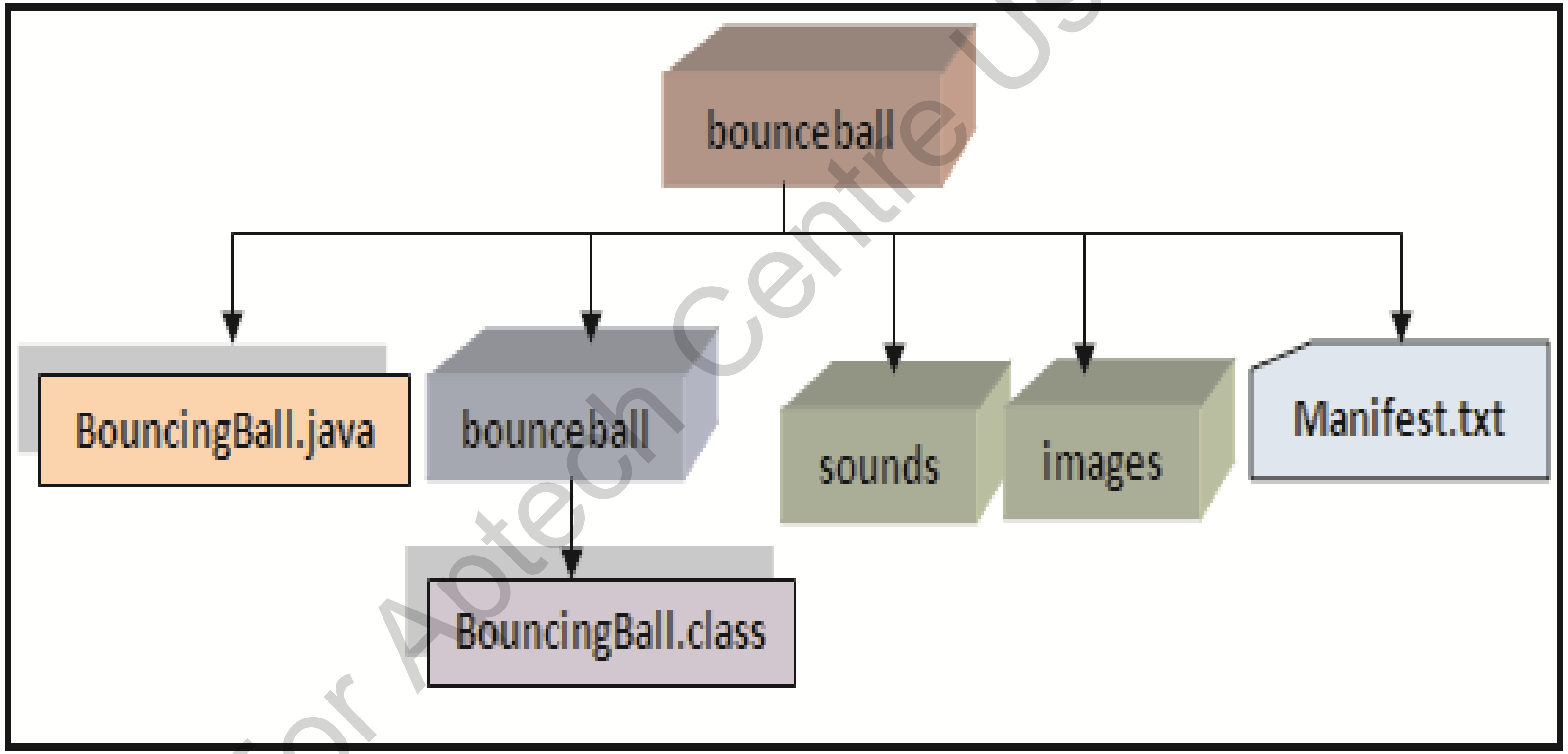


Creating .jar Files for Deployment 1-9

Security	• The `.jar` file can be digitally signed so that only those users who recognize your signature can optionally grant the software security privileges.
Decrease in Download Time	• The source files bundled in a `.jar` file can be downloaded to a browser in a single HTTP transaction without having to open a new connection for each file.
File Compression	• The `.jar` format compresses the files for efficient storage.
Packaging for Extensions	• The extension framework in Java allows adding additional functionality to the Java core platform.
Package Sealing	• Java provides an option to seal the packages stored in the `.jar` files so that the packages can enforce version consistency.
Package Versioning	• A `.jar` file can also store additional information about the files, such as vendor and version information.
Portability	• This enables the user to use them for tasks such as lossless data compression, decompression, archiving, and archive unpacking.

Creating .jar Files for Deployment 2-9

- It is advisable to use versioning information in the manifest file instead of creation time, to control versions of a `.jar` file.
- Consider the following files of a simple **`BouncingBall`** game application as shown in the following figure:



Creating .jar Files for Deployment 3-9

- To create a `.jar` of the application using command line, perform the following steps:

1
- Create the directory structure as shown in the earlier figure.

2
- Create a text file with the code depicted in the following code snippet:

```
package bounceball;
public class BouncingBall {

  /**
   * @param args the command line arguments
   */
  public static void main(String[] args) {
    System.out.println("This is the bouncing ball game");
  }
}
```

3
- Save the file as **`BouncingBall.java`** in the source package **`bounceball`**.

Creating .jar Files for Deployment 4-9

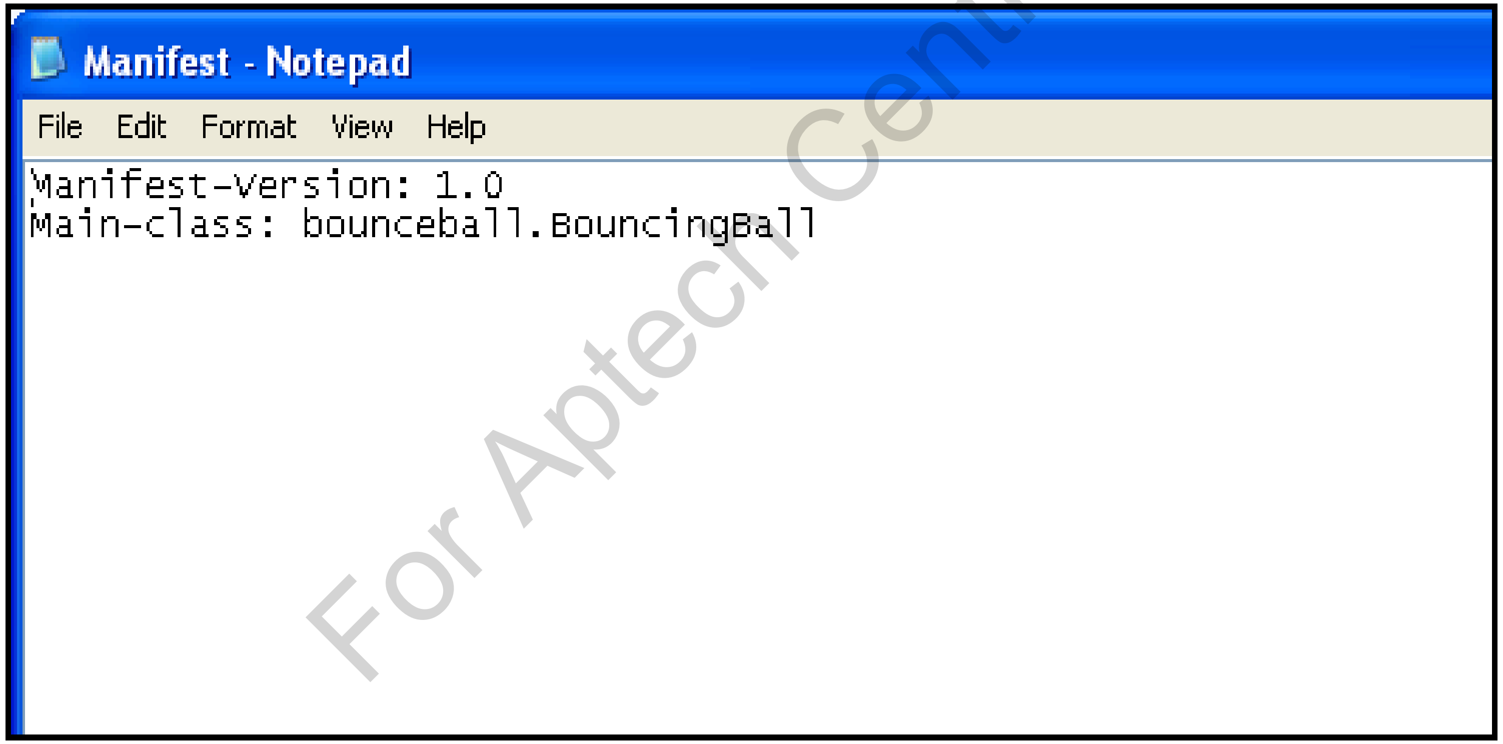
4

- Compile the `.java` file at command prompt by writing the following command:

```
javac -d . BouncingBall.java
```

5

- Create a text file with the `Main-class` attribute as shown in the following figure:



```
Manifest-Version: 1.0
Main-class: bounceball.BouncingBall
```

Creating .jar Files for Deployment 5-9

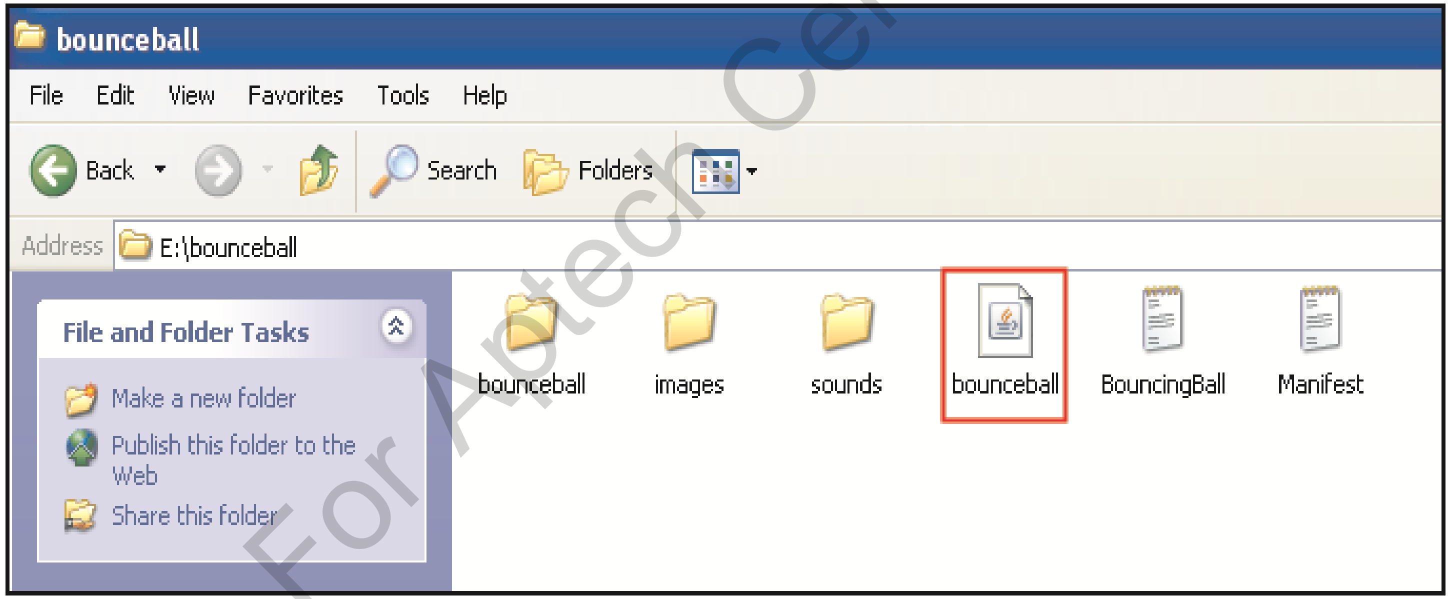
6

- Save the file as **`Manifiest.txt`** in the source **`bounceball`** folder.

7

- To package the application in a single `.jar` named **`BouncingBall.jar`**, write the following command:

```
jar cvmf Manifest.txt bounceball.jar
   bounceball/BouncingBall.class sounds images
```



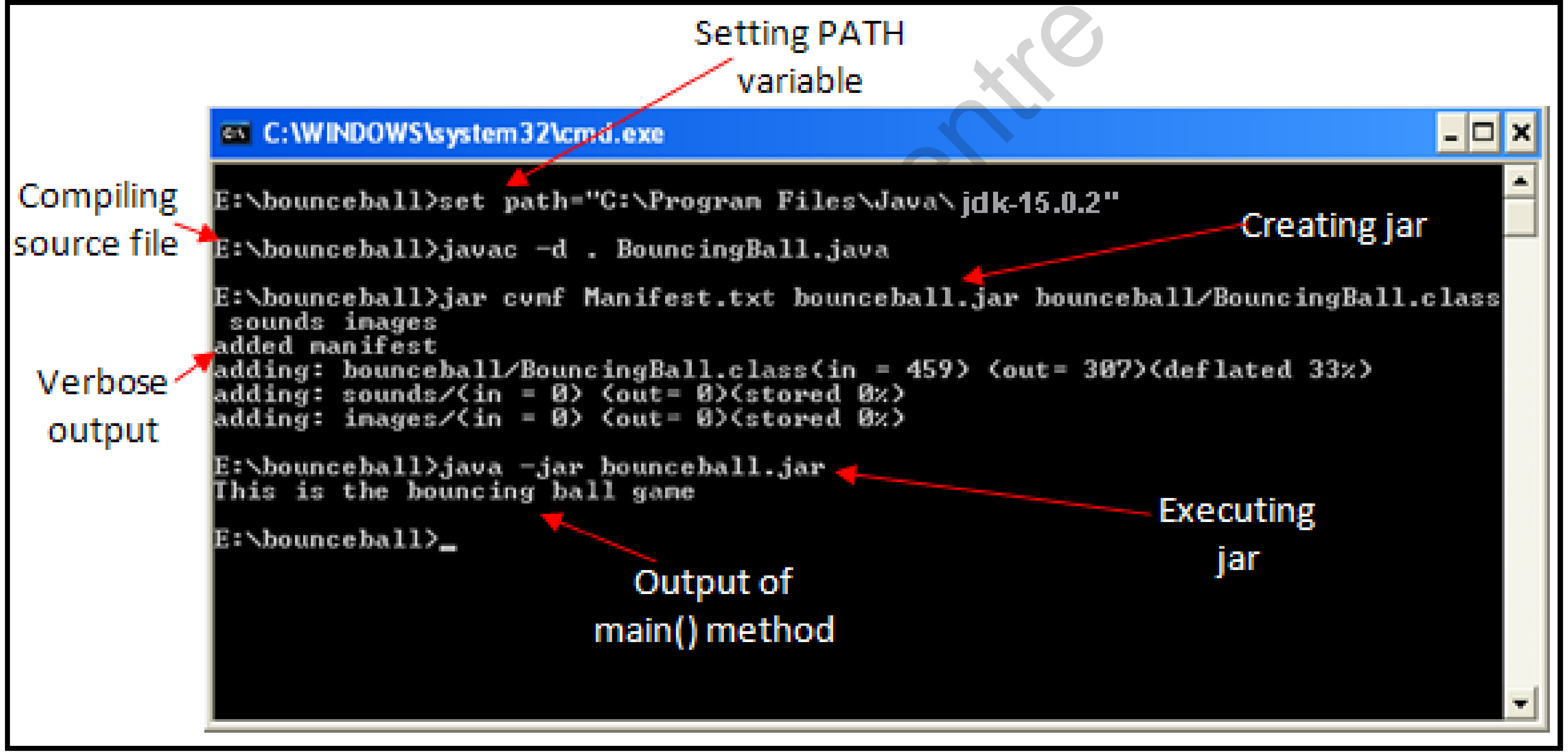
Creating .jar Files for Deployment 6-9

8

- To execute the `.jar` file at command prompt, type the following command:

```
java -jar bounceball.jar
```

- The command will execute the `main()` method of the class of the `.jar` file and print the following output:



```
E:\bounceball>set path="C:\Program Files\Java\jdk-15.0.2"

E:\bounceball>javac -d . BouncingBall.java

E:\bounceball>jar cvmf Manifest.txt bounceball.jar bounceball/BouncingBall.class
 sounds images
added manifest
adding: bounceball/BouncingBall.class(in = 459) (out= 307)(deflated 33%)
adding: sounds/(in = 0) (out= 0)(stored 0%)
adding: images/(in = 0) (out= 0)(stored 0%)

E:\bounceball>java -jar bounceball.jar
This is the bouncing ball game

E:\bounceball>_
```

Creating .jar Files for Deployment 7-9

Task	Command
To create a .jar file	jar cf jar-file-name input-file-name (s)
To view contents of a .jar file	jar tf jar-file-name
To extract contents of a .jar file	jar xf jar-file-name
To run the application package into the .jar file. **Manifest.txt** file is required with Main-class header attribute	java -jar jar-file-name

- To create a `.jar` file of the application using NetBeans IDE, perform the following steps:

1
- Create a new package **bounceball** in the **Session6** application.

2
- Create a new java class named **BouncingBall.java** within the **bounceball** package.

3
- Type the code depicted in the following code snippet in the **BouncingBall** class:

Creating .jar Files for Deployment 8-9

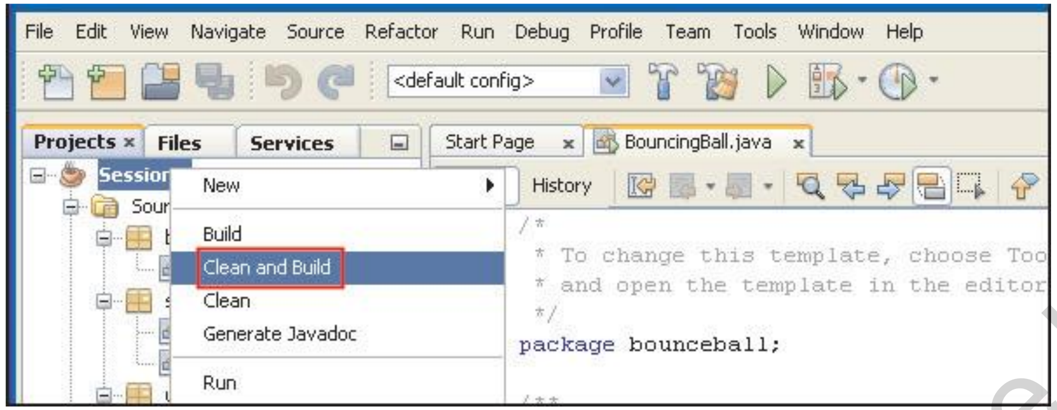
```
package bounceball;
  public class BouncingBall {
  /**
   * @param args the command line arguments
   */
  public static void main(String[] args)
  {
    System.out.println("This is the bouncing ball game");
  }
}
```

4
- Set **BouncingBall.java** as the main class in the **Run** properties of the application.

5
- Run the application by clicking the **Run** icon on the toolbar.

6
- To create the `.jar` file, select **Clean and Build** option as shown in the following figure:

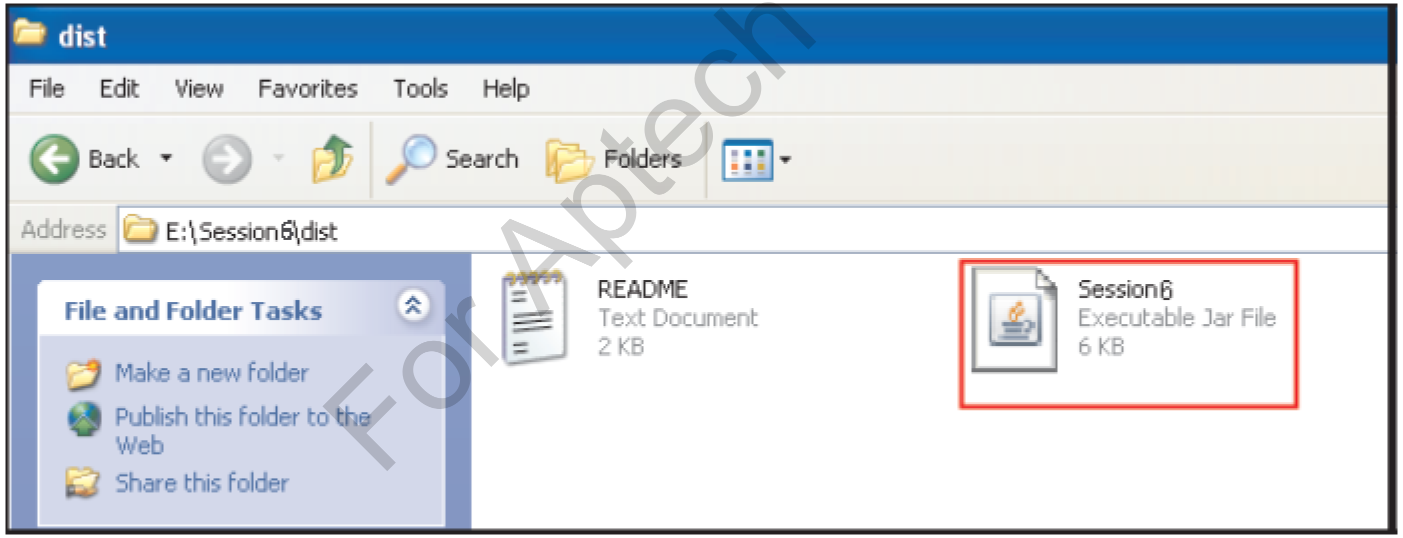
Creating .jar Files for Deployment 9-9



```
/*
 * To change this template, choose Too
 * and open the template in the editor
 */
package bounceball;
```

- The IDE will build the application and generate the `.jar` file.

- The **Clean and Build** command creates a new dist folder into the application folder and places the `.jar` file into it as shown in the following figure:



Summary

- Field and method modifiers are used to identify fields and methods that have controlled access to users.
- The volatile modifier allows the content of a variable to be synchronized across all running threads.
- A thread is an independent path of execution within a program.
- The native modifier indicates that the implementation of the method is in a language other than Java such as C or C++.
- The transient modifier can only be used with instance variables. It informs the JVM not to store the variable when the object, in which it is declared, is serialized.
- The final modifier is used when modification of a class or data member is to be restricted.
- Class variables are also known as static variables and there exists only one copy of that variable for all objects.
- A package is a namespace that groups related classes and interfaces and organizes them as a unit.
- All the source files of a Java application are bundled into a single archive file called the Java Archive (JAR).