Packages:
—--------
Def:

1. Package is the collection of related classes and interfaces as a single unit.
2. Package is a folder that contains .class files representing related classes and interfaces.

Advantages:

1. Modularity:
   In Java application development, we will divide the complete application source into the number of logical units called “Modules”, where to represent modules in java applications we have to use packages.

2. Abstraction:
   In Java applications, if we declare classes and interfaces in a package then that classes and interfaces are hidden by the package , by default the classes and interfaces of a particular package are not available to outside of the package, so Packages are able to provide abstraction to their internal declarations.

3. Security:
   In Java applications, packages are able to hide their internal declarations, so packages are able to provide security for their internal declarations.

4. Shareability:
   If we declare a package with classes and interfaces then we are able to share that single copy of that package to multiple java applications at a time, so packages are able to provide shareability.

5. Reusability:
   If we define a package one time that we can reuse in any number of times as per the application requirements.

Types:
There are two types of packages in java applications.

1. Predefined Packages
2. User defined packages

Predefined Packages:
—-------------------
These packages are defined by the Java programming Language and provided along with Java software.

Java has provided the complete predefined packages in the rt.jar file at “C:\Java\jdk1.8.0_202\jre\lib\rt.jar”.

EX1: java.lang package:
It is a default package in java applications, it is not required to import this package to use classes and interfaces from this package.

This package contains all the fundamental classes and interfaces which are required to prepare basic java applications.

EX: String, StringBuffer, StringBuilder,....
     Byte, Short, Integer, Float, Character,.....
     Exception, Throwable, NullPointerException, ArithmeticException,.....
     Object, System, Math,.......

EX2: java.io package:
This package contains all the classes and interfaces which are required to perform input and output operations in java applications.

EX: InputStream, ByteArrayInputStream, FileInputStream, DataInputStream,....
    OutputStream, ByteArrayOutputStream, FileOutputStream,........
    Reader, FileReader, BufferedReader, InputStreamReader,.....
    Writer, FileWriter, BufferedWriter, PrintWriter,.......
    Serializable, Externalizable,........

Ex3: java.util package:
This package contains all the classes and interfaces which are required to represent data structures.

EX: List, ArrayList, Vector, Stack, LinkedList,...
     Set, HashSet, LinkedHashSet,SortedSet, navigableSet, TreeSet,....
     Queue, PriorityQueue, LinkedPriorityQueue,......
     Map, HashMap, LinkedHashMap, IdentityHashMap, WeakHashMap, SortedMap,
     NavigableMap, TreeMap, Properties, Hashtable,.......
     Comparable, Collection, Collections,.......
     Enumeration, Iterator, Listiterator,....

EX4: java.awt package:
This package contains all the predefined classes and interfaces which are required to prepare GUI applications.

EX: Label, TextField, Button, Checkbox, CheckboxList, Menu, TextArea, ChoisBox, List,.....
Window, Panel, Applet,.....
FlowLayout, BorderLayout, GridLayout, GridbagLayout,....

EX5: javax.swing package:
This package contains all the predefined classes and interfaces which are required to prepare GUI applications more powerful than AWT GUI applications.

EX: JWindow, JLabel, JTextField, JPasswordField, JCheckBox, JRadioButton, JList, JComboBox, JTextArea, JTree, JColorChooser, JFileCHooser,.....

EX6: java.sql package:
This package contains all the predefined classes and interfaces which are required to prepare JDBC applications.

EX: Connection, Driver, DriverManager, Statement, PreparedStatement, CallableStatement, ResultSet, DatabaseMetaData, resultSetMetaData,...

EX7: java.net package:
This package contains all the predefined classes and interfaces which are required to prepare Distributed applications by using Socket programming.

EX: Socket, ServerSocket, URL, URI, URLConnection,.......

EX8: java.rmi package:
This package contains all the predefined classes and interfaces which are required to prepare Distributed applications by using Remote Method invocation.

EX: Naming, RMIRegistry, Remote,.......

EX9: java.text package:
This package contains all the classes and interfaces which are required to prepare Internationalization[I18N] applications.

EX: NumberFormat, DateFormat, ResourceBundle,......

User Defined Packages:
—----------------------
These packages are defined by the developers as per their application requirements.

To declare user defined packages we have to use the “package” statement.

Package packageName;

Where packageName may be a single name or the combination of the parent package name and child package name with . separator.

EX: package p;
     package p1.p2.p3;

If we want to use a package declaration statement in a java file then we must use the following conditions.

1. The Package Declaration statement must be the first statement in the java file.
2. Package names must be unique, they must not be shareable, they must not be reusable.

Q)Is it possible to provide more than one package declaration statement in a single java file?
—--
Ans:
—---
No, It is not possible to provide more than one package declaration statement in a single java file.

EX: abc.java
package p1; —-----> Valid
package p2; —-----> Invalid
package p3; —-----> Invalid

To provide package names , JAVA has provided a suggestion like to include our company domain name in reverse in the package name.

EX:
package com.durgasoft.icici.transactions.deposit;

com.durgasoft —-----> Company Domain name in reverse.

icici —-----> Client /Project name
transaction —----> Module name
deposit —-----> Sub module name

If we want to use classes and interfaces of a particular package in the present java file then we have to import that package in the present java file.

To import a package into the present java file we have to use “import” statement.

Syntax#1: import packageName.*;
It is able to import all the classes and interfaces of a specified package into the present java file.
EX: import java.io.*;
     import java.util.*;

Syntax#2: import packageName.memberName;
It is able to import only the specified member from the specified package.
EX: import java.io.BufferedReader;
     import java.util.ArrayList;

Note: In a java file, we are able to provide at most one package declaration statement , but we are able to provide any number of import statements.

Ex: abc.java
package p1; —----------> Valid
package p2; —----------> Invalid
package p3; —----------> Invalid
import java.io.*; —-----> Valid
import java.util.*; —---> Valid
import java.sql.*; —----> Valid

Q)Is it possible to use classes and interfaces of a particular package into the present java file without importing the respective package?
—----------

Ans:
—---
Yes, it is possible to use classes and interfaces of a particular package without importing the respective package but we must use “Fully Qualified Names” of the classes and interfaces.

Note: Providing classes names and interfaces names along with their package names is called Fully Qualified name.
EX: java.io.BufferedReader
java.util.ArrayList

A Java program with import statement:
—----------
```
import java.io.*;

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
```

A Java program without import statement:
—----------

```
java.io.BufferedReader br = new java.io.BufferedReader(new
java.io.InputStreamReader(System.in));
```

Note: In Java applications, it is always suggestible to use import statements when compared with fully qualified names.

Note: In the case of import statements, it is always suggestible to import individual members when compared with '*' notation.
EX: import java.io.*; —---> Not Suggestible
import java.io.BufferedReader; —-----> Suggestible
import java.io.InputStreamReader; —---> Suggestible

Q)In which situations we must use fully qualified names to the classes in java applications?
—----------
Ans:
—---
If we have the same class in more than one package and if we use that class in the present java file then there is confusion from which package the provided class is accessed. In this context, to resolve the confusion we need fully qualified names.

EX: In java, Date is a class available in java.util package and in java.sql package.

java.util.Date is for representing Current System date.
java.sql.Date is for representing a date value in the databases.

EX:

```
import java.util.*;
import java.sql.*;
public class Test{
      public static void main(String[] args){
            Date dt1 = new Date();
            System.out.println(dt1);

            Date dt2 = Date.valueOf("1996-12-12");
            System.out.println(dt2);
      }
}
```

```
D:\java6>javac Test.java
Test.java:5: error: reference to Date is ambiguous
                Date dt1 = new Date();
                ^
  both class java.sql.Date in java.sql and class java.util.Date in java.util
match
Test.java:5: error: reference to Date is ambiguous
                Date dt1 = new Date();
                           ^
  both class java.sql.Date in java.sql and class java.util.Date in java.util
match
Test.java:8: error: reference to Date is ambiguous
                Date dt2 = new Date.valuerOf("1996-12-12");
                ^
  both class java.sql.Date in java.sql and class java.util.Date in java.util
match
Test.java:8: error: reference to Date is ambiguous
                Date dt2 = new Date.valueOf("1996-12-12");
                           ^
  both class java.sql.Date in java.sql and class java.util.Date in java.util
match
4 errors

D:\java6>
```

In the above program to resolve the confusion we have to use Fully qualified names.

EX:

```
import java.util.*;
import java.sql.*;
public class Test{
      public static void main(String[] args){
            java.util.Date dt1 = new java.util.Date();
            System.out.println(dt1);

            java.sql.Date dt2 = java.sql.Date.valueOf("1996-12-12");
            System.out.println(dt2);
      }
}
```

```
D:\java6>javac Test.java

D:\java6>java Test
Sat Apr 01 20:19:03 IST 2023
1996-12-12

D:\java6>
```

EX1:

```
E:\abc
|---Employee.java[package name: com.durgasoft.emp]
```

Employee.java

```
package com.durgasoft.emp;
public class Employee{

      private int eno;
      private String ename;
      private float esal;
      private String eaddr;

      public Employee(int eno, String ename, float esal, String eaddr){
            this.eno = eno;
            this.ename = ename;
            this.esal = esal;
            this.eaddr = eaddr;
      }
```

```
	public void getEmployeeDetails(){
		System.out.println("Employee Details");
		System.out.println("----------------------");
		System.out.println("Employee Number    : "+eno);
		System.out.println("Employee Name      : "+ename);
		System.out.println("Employee Salary    : "+esal);
		System.out.println("Employee Address   : "+eaddr);
	}
}
```

Compile Employee.java and send the generated Employee.class file to C:\xyz location

```
E:\abc>javac -d C:\xyz Employee.java
```

```
C:\xyz
|---com
	|----durgasoft
		|----emp
			|----Employee.class
```

```
D:\java6\packages\app01
|----Test.java
|----Test.class
```

```
Test.java
import com.durgasoft.emp.Employee;
public class Test{
	public static void main(String[] args){
		Employee emp = new Employee(111, "Durga", 5000, "Hyd");
		emp.getEmployeeDetails();
	}
}
```

If we want to compile the above java file then we must set the “classpath” environment variable to the location where the “com.durgasoft.emp” package exists.

```
D:\java6\packages\app01>set classpath=C:\xyz;

D:\java6\packages\app01>javac Test.java

D:\java6\packages\app01>java Test
Employee Details
----------------------
Employee Number    : 111
Employee Name      : Durga
Employee Salary    : 5000.0
Employee Address   : Hyd

D:\java6\packages\app01>
```

EX2:

—---

E:\abc

|---Student.java[package Name: com.durgasoft.std]

Student.java

```
package com.durgasoft.std;
public class Student{

      private String sid;
      private String sname;
      private String saddr;

      public Student(String sid, String sname, String saddr){
            this.sid = sid;
            this.sname = sname;
            this.saddr = saddr;
      }

      public void getStudentDetails(){
            System.out.println("Student Details");
            System.out.println("----------------------");
            System.out.println("Student Id      : "+sid);
            System.out.println("Student Name    : "+sname);
            System.out.println("Student Address : "+saddr);
      }

}
```

Compile Student.java file and send the generated Student.class file to C:\xyz location.

```
E:\abc>javac -d C:\xyz Student.java

E:\abc>
```

```
C:\xyz
|----com
      |----durgasoft
            |-----std
                  |-----Student.class
```

```
C:\abc
|---Customer.java[Package Name: com.durgasoft.cust]
```

Customer.java

```
package com.durgasoft.cust;
public class Customer{

      private String cid;
      private String cname;
      private String caddr;

      public Customer(String cid, String cname, String caddr){
            this.cid = cid;
            this.cname = cname;
            this.caddr = caddr;
      }

      public void getCustomerDetails(){
            System.out.println("Customer Details");
            System.out.println("-----------------------");
            System.out.println("Customer Id       : "+cid);
            System.out.println("Customer Name     : "+cname);
            System.out.println("Customer Address  : "+caddr);
      }
```

```
}
```

Compile Customer.java file and send the generated Customer.class file to E:\xyz location .

```
C:\abc>javac -d E:\xyz Customer.java

C:\abc>
```

```
E:\xyz
|----com
      |----durgasoft
            |----cust
                  |----Customer.class
```

```
D:\java6\packages\app02
|----Test.java[package Name: com.durgasoft.test]
```

Compile Test.java and generate Test.class file at the same current location with the package respective folder structure.

```
D:\java6\packages\app02
|---com
      |----durgasoft
            |----test
                  |---Test.class
```

Test.java

```
package com.durgasoft.test;
import com.durgasoft.std.*;
import com.durgasoft.cust.*;
public class Test{
      public static void main(String[] args){
            Student std = new Student("S-111", "AAA", "Hyd");
            std.getStudentDetails();
            System.out.println();

            Customer cust = new Customer("C-111", "BBB", "Pune");
```

```
            cust.getCustomerDetails();
      }
}


D:\java6\packages\app02>set classpath=C:\xyz;E:\xyz;

D:\java6\packages\app02>javac -d . Test.java

D:\java6\packages\app02>java com.durgasoft.test.Test
Student Details
----------------------
Student Id      : S-111
Student Name    : AAA
Student Address : Hyd

Customer Details
----------------------
Customer Id       : C-111
Customer Name     : BBB
Customer Address  : Pune

D:\java6\packages\app02>set classpath=%classpath%;.;

D:\java6\packages\app02>java com.durgasoft.test.Test
Student Details
----------------------
Student Id      : S-111
Student Name    : AAA
Student Address : Hyd

Customer Details
----------------------
Customer Id       : C-111
Customer Name     : BBB
Customer Address  : Pune

D:\java6\packages\app02>
```

JAR Files In Java:
—----------

JAR: Java Archive .

In Java applications, JAR file is the collection of packages that contains .class files.

To create jar files we have to use the following command.

```
jar -cvf fileName.jar *
```

C– Create
V– Verbose
F– File

To get the content from the jar file we have to use the following command on command prompt.

```
jar -xvf fileName.jar
```

X– Extract
V– Verbose
F– File

If we want to access the content of a jar file in the present java file then we have to set the classpath environment variable to the respective jar file.

EX: set classpath=E:\abc\xyz\app.jar;

In general, in java applications we need jar files in the following requirements.

1. To simplify the delivery process of the software applications.
2. To simplify the uploading and downloading projects from the websites.
3. To Simplify the project transformation from one machine to another machine.
4. To simplify the deployment process of the software applications into the servers.

EX4:
E:\abc
|---Account.java[package Name: com.durgasoft.icici.accounts]

Account.java

```
package com.durgasoft.icici.accounts;
public class Account{

	private String accNo;
	private String accHolderName;
	private String accType;
	private int balance;

	public Account(String accNo, String accHolderName, String accType, int
balance){
		this.accNo = accNo;
		this.accHolderName = accHolderName;
		this.accType = accType;
		this.balance = balance;
	}

	public void getAccountDetails(){
		System.out.println("Account Details");
		System.out.println("-----------------------");
		System.out.println("Account Number      : "+accNo);
		System.out.println("Account Holder Name : "+accHolderName);
		System.out.println("Account Type        : "+accType);
		System.out.println("Account Balance     : "+balance);
	}

}
```

Compile the Account.java file and send the generated .class file to C:\abc location

```
E:\abc>javac -d C:\abc Account.java

E:\abc>
```

```
C:\abc
|---com
      |---durgasoft
            |----icici
                  |----accounts
                        |-----Account.class
```

Prepare a jar file with the name icici_account.jar for the com.durgasoft.icici.accounts.Account.class file and move this jar file to D:\abc location.

```
C:\abc>jar -cvf icici_account.jar *
```

We copy this jar file to D:\abc.

```
D:\abc
|---icici_account.jar

D:\java6\packages\app03
|---Test.java
|---Test.class
```

Test.java

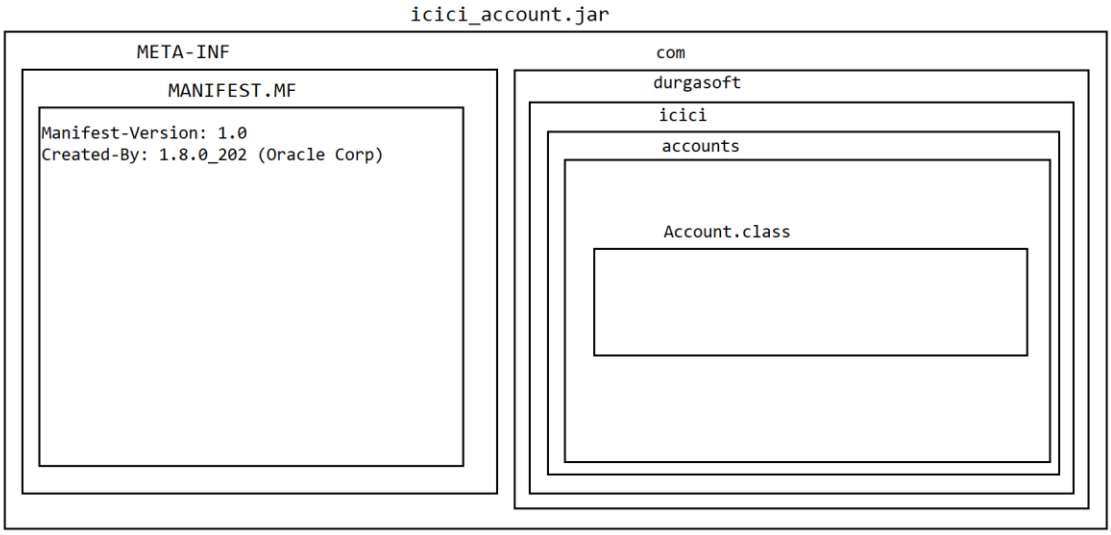
```
import com.durgasoft.icici.accounts.*;
public class Test{
      public static void main(String[] args){
            Account account = new Account("abc123", "Durga", "Savings",
50000);
            account.getAccountDetails();
      }
}
```

```
D:\java6\packages\app03>set classpath=D:\abc\icici_account.jar;

D:\java6\packages\app03>javac Test.java

D:\java6\packages\app03>java Test
Account Details
------------------------
Account Number      : abc123
Account Holder Name : Durga
Account Type        : Savings
```

```
Account Balance     : 50000

D:\java6\packages\app03>
```

MANIFEST File in Jar:
—--------------------
MANIFEST.MF is a file created by the jar command at the time of creating a jar file under META-INF folder.

MANIFEST.MF file is able to provide information about the jar file in the form of Name-value pairs.



Executable Jar File:
—---------------------
If any jar file contains the main class and main() method along with other packages then that jar file is an executable jar file.

Steps:

1. Prepare Java application and compile java files.

```
Test.java
import java.awt.*;
class LogoFrame extends Frame{
      public LogoFrame(){
            this.setVisible(true);
            this.setSize(500, 500);
            this.setBackground(Color.green);
            this.setTitle("Logo Frame");
```

```
        }
        public void paint(Graphics g){
                Font font = new Font("arial", Font.BOLD, 30);
                g.setFont(font);
                g.drawString("DURGA SOFTWARE SOLUTIONS", 100,100);
        }
}
class Test{
        public static void main(String[] args){
                LogoFrame logo = new LogoFrame();
        }
}
```

```
D:\java6\packages\app04>javac Test.java

D:\java6\packages\app04>
```

2. Create a text file with Main Class Details:

```
abc.txt
Main-Class: Test
```

3. Create a jar file with the user defined manifest data.

```
D:\java6\packages\app04>jar -cvfm logo.jar abc.txt *
added manifest
adding: abc.txt(in = 18) (out= 20)(deflated -11%)
adding: LogoFrame.class(in = 752) (out= 502)(deflated 33%)
adding: Test.class(in = 285) (out= 221)(deflated 22%)
adding: Test.java(in = 461) (out= 278)(deflated 39%)
adding: Test.java.bak(in = 458) (out= 277)(deflated 39%)

D:\java6\packages\app04>
```

4. Execute jar file:

```
D:\java6\packages\app04>java -jar logo.jar
```

When we execute the jar file, JVM will perform the following actions.

1. JVM will go inside the jar file and search for MANIFEST.MF file in META-INF folder.
2. IN MANIFEST.MF file, JVM will search for the Main-Class attribute value that is the Main class name.

3. When JVM identifies the main class name , JVM will search for the main class in the jar file and JVM will execute the main class.

5. To simplify the execution of jar file we have to create a batch file.
   logo.bat
   java -jar logo.jar

6. Copy logo.jar and logo.bat file on Desktop and double click on bat file