Date 19/9/16
Page

Unit III

X Interfaces, Packages And String Handling

Introduction to interfaces

While discussing the concept of inheritance, we have seen that java does not support multiple inheritance. A large number of real applications requires the use of multiple inheritance. Where we can inherit methods and properties from several distinct classes. Java provides an alternate approach which overcomes such difficulty, the approach is known as interfaces to support the concept of multiple inheritance.

Although a java class can not be a sub class of more than one super class, it can implement more than 1 interface by enabling us to create classes which builds upon other classes without the problems created by multiple inheritance.

* Defining an Interfaces

"An interface is a basically a kind of a class. like classes, interfaces contain methods and variables but with a major difference. The difference is that, interface define only abstract method and final fields. This means that interfaces do not specify any code to implement these methods and the data fields which contains only constants.

Therefore, it is the responsibility of the class that implements an interface to define the code for the implementation of the methods. The syntax of defining an interface is very similar to that for defining a class.

The general syntax for defining an interface is

```
Interface Interfacename
{
    variables declaration;
    methods declaration;
}
```

Page 23/9

Single Inheritance

```
class Room
{
    int length;
    int breadth;

    Room(int x, int y)
    {
        length = x;
        breadth = y;
    }

    int area()
    {
        return(length * breadth);
    }
}

class BedRoom extends Room
{
    int height;

    BedRoom(int x, int y, int z)
    {
        super(x,y);
        height = z;
    }
```

```
int volume()
{
    return (length * breadth * Height);
}
}

class InherTest
{
  public static void main(String args[])
  {
    BedRoom room1 = new BedRoom
                    (14, 12, 10);

    int area1 = room1.area();
    int volume1 = room1.volume();

    System.out.println("Area1 = "+area1);
    System.out.println("volume = "+volume1);
  }
}
```

Above program will be generate output as -

```
Area 1 = 168
volume = 1680
```

Date 28/9/2018

In the above syntax, interface is the keyword and interface name is any valid name of an interface like a class interface can also contains declaration of variables as well as methods. Note that all the variables declared inside an interface are constants while the methods declaration will only contains a list of methods without any definitions.

eg

```
interface Area
{
    static final float pi=3.14;
    float compute(float x, float y);
    void display();
}
```

In the above example, Area is nothing but it is an name of an interface which contains one variable and two methods. The Area interface will contains pi as a float variable having initial value as 3.14. This interface also contains

Compute method having returntype as Float. and also contains two Float parameters. There is also a display() method included inside an area Interface. Note that the code for the method is not included in the interface and the method declarations are simply ends with a semicolon(;). The class which can impliment this Interface must define the codes for the specific methods.

* Extending Interfaces

Like classes, Interfaces can also be extended. That is an Interface can be sub interfaced from other Interfaces. The new sub Interface will inherit all the members of the sub Interface in the manner similar to sub classes. This is achieved by using the keyword extends.

The general syntax for extending an Interfaces are

```
Interface name2 extends name1
{
    body of name2
}
```

In the above syntax, Interface is the keyword, name2 is an name of new Interface (sub Interface). which is extended from existing Interface, extends is a keyword which represents one Interface can be extended from existing Interface and name1 is the name of already existing Interface.

We can design the code regarding with the new sub interface. We can generally implement all the constants in one Interface and the methods in the other. This will enables us to use the constants in the classes where the methods are not required.

eg

```
Interface ItemCost
{
    int code = 1007;
    String name = "Perk";
}
```

```
interface Item extends ItemCost
{
    void display();
}
```

In the above example, the interface ItemCost is an original interface which contains two members namely code and name belonging to int and string type. The interface Item is an new sub-interface which would inherit both the constants code and name into it. Note that the variables name and code are declared like simple variables. It is allowed because all the variables in an interface are treated as constants although the keywords like final and static are not placed before them.

We can also combine several interfaces together into a single interface.

* Implementing Interfaces

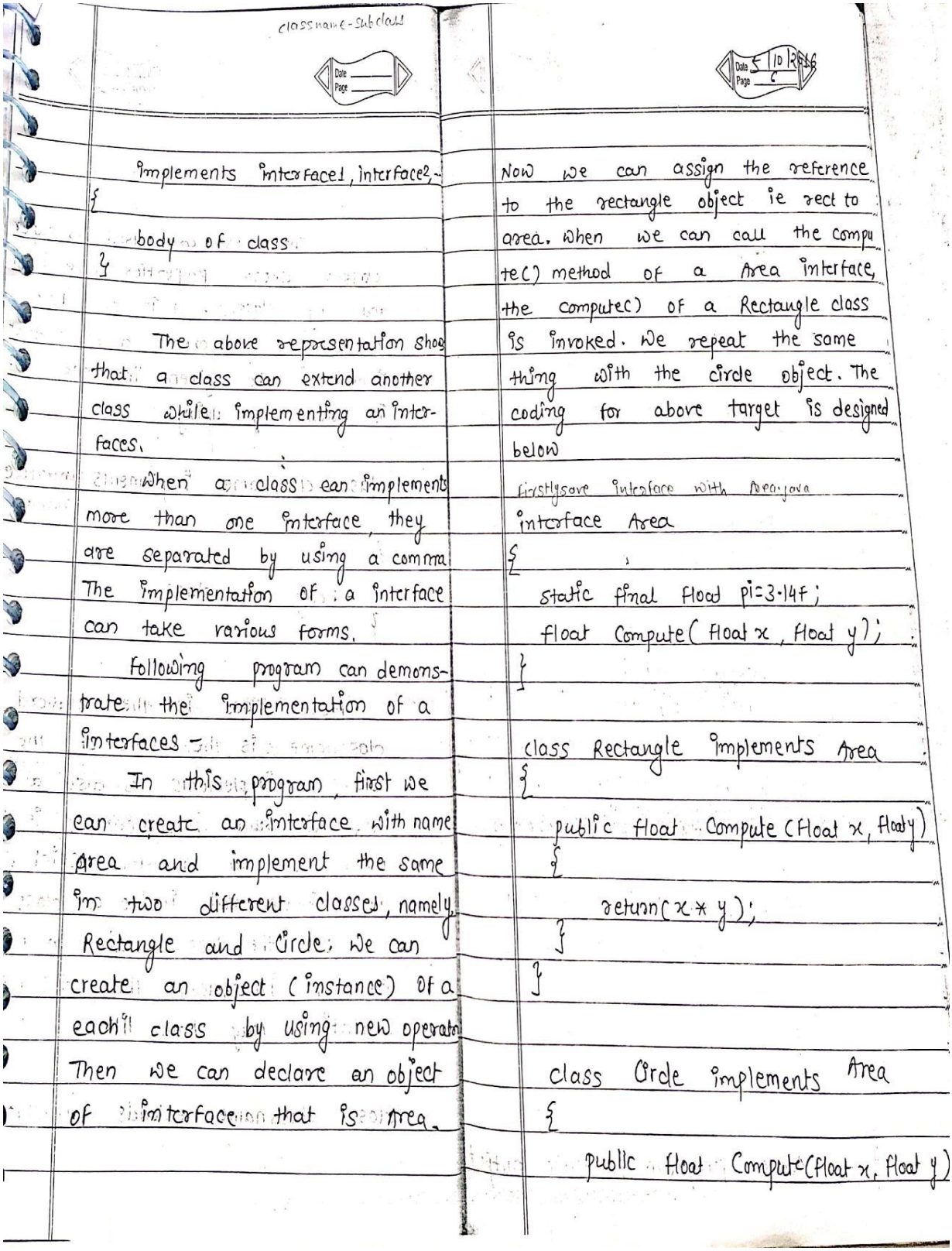
Interfaces are used as super classes whose properties are inherited by classes. It is therefore necessary to create a class that inherits the given interface. This is done by using

```
class classname implements Interfacename
{
    body of class
}
```

Here, class is the keyword, classname is the name of the class, implements is also a keyword and interfacename is the name of an interface which we are implemented inside a class.

If the class contains n number of interfaces then, the representation may look like

```
class classname extends superclass-
```



classname-Subclass

Implements interface1, interface2,..
{
body of class
}

The above representation shows that a class can extend another class while implementing an interfaces.

When a class can implements more than one interface, they are separated by using a comma. The implementation of a interface can take various forms.

Following program can demonstrate the implementation of a interfaces -

In this program, first we can create an interface with name Area and implement the same in two different classes, namely Rectangle and Circle. We can create an object (instance) of a each class by using new operator. Then we can declare an object of interface that is Area.

Now we can assign the reference to the rectangle object ie rect to area. When we can call the compute() method of a Area interface, the compute() of a Rectangle class is invoked. We repeat the same thing with the circle object. The coding for above target is designed below

firstly save interface with Area.java

```
interface Area
{
  static final float pi=3.14f;
  float Compute(float x, float y);
}

class Rectangle implements Area
{
  public float Compute(float x, float y)
  {
    return(x*y);
  }
}

class Circle implements Area
{
  public float Compute(float x, float y)
```

```
    {
        return (Pi * x * x);
    }
}

class InterfaceTest
{
    public static void main(String args[])
    {
        Rectangle rect = new Rectangle();
        Circle cir = new Circle();

        Area area;
        area = rect;
        area = cir;

        System.out.println("Area of Rectangle
            " + area.compute(10,20));
        area = cir;
        System.out.println("Area of Circle=" +
            area.compute(25.0,0));
    }
}
```

Above program generate output as-

Area of Rectangle = 200.0

Area of Circle =

* Packages

Introduction

Java accomplish a concept of class libraries which is a way of achieving the reusability of a classes. The concept is known as packages.

"An packages are java's way of grouping a variety of classes and/or interfaces together." The grouping is usually done according to the functionality. Generally packages acts as containers for classes. following are some benefits for using packages into our classes.

1) The classes contained in the packagy of a other programs can be easily reused.

2) In packages, classes can be unique compared with classes in other packages. That is the two classes in two different packages can have the

same name. They may be referred by their names and the class name

3) Packages provide a way to hide the classes for preventing other programs or packages from accessing classes for internal use only.

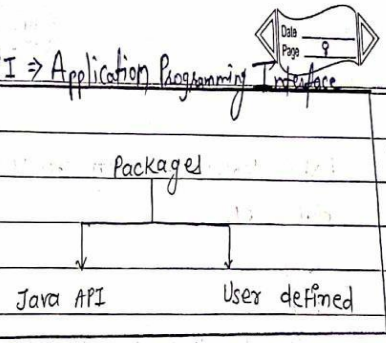
4) Packages also provide a way for separating design from coding ie. first we can design classes and then we can implement the java code needed for the methods. It is possible to change the implementation of a any method without affecting the rest of the design.

* Types of packages

We have seen that we can build our own classes for handling our data and use existing class libraries for designing user interfaces. like that, java packages are classified into two types

1) Java API Packages
2) User defined packages

API ⇒ Application Programming Interface



1) Java API packages

Java application programming interface packages provides a large number of classes grouped into different packages according to its functionality.

Most of the time, we used packages available with java API following are the well known, frequently used java API packages

1. java.lang →

java.lang support classes. These are the class which java compiler itse. If uses and, therefore they are automatically imported.

This package includes classes for premitive types, strings, math functions thread and exception

java.util →

language utility classes such as vecto

hash tables, random number numbers, date etc

java.io →
Input /output support classes they provides facilities for the input & output of data.

java.awt →
Set of classes used for implementing graphical user interface. They include classes for windows buttons, buttons, lists menus and so on

java.net →
classes for networking. They include classes for communicating with local computers as well as Internet server

java.applet →
classes for creating & implementing applets

2) User defined packages -
As the name suggest, the packages which are designed and created by user for their convinience & then those packages are called as user defined packages. for creating our own packages we must first declare the name of the package by using the package keyword followed by a valid name of the package. This must be the first statement in java source file except for comments and white spaces, then we define a class with valid name just we normally defines a class.

The general syntax for creating a package is

```
package package-name;
```

The general way for representing a class inside a package is

```
package mypackage;
public class Myclass
{
```

--- (body of class)

}

In above example, mypackage is nothing but the valid name of the package which contains a class namely Myclass followed with public keyword. The class Myclass is now considered a part of mypackage. After creating the class, it should be saved as a file with name myclass.java and located in a directory named mypackage. When the source file is compiled, java will create a .class file and store it in the same directory.

Remember that, the dot .class file must be located in a directory which has the same name as the package and the directory should be a sub directory where classes that will import the package are located.

For creating our own packages, we can perform following basic steps:

1) Declare the package at the begining of a file by using a valid package name.
2) Define a class inside the package and declare it as a public
3) Create a sub directory under the directory where the main source files are stored.
4) Store the listing as the name with class-name.java file in the sub directory created.
5) Compile the file which creates .class file in the sub-directory.

Remember that the sub-directory name must match with the name of the package. If we want to create the hierarchy of packages specifying multiple names then this

can be done by separating each package name with dots.
for eg-

```
package firstpackage.Secondpackage;
```

This approach allows us to group related classes into a package and then group related packages into a main package.

* Accessing the package

As discussed earlier, a java system packages can be accessed by using an import statement within our programs. We can use an import statement when there are many references to a particular package or the package name is too long. The same approach can be used to access the user defined packages where the import statement can be used to search a list of packages for a particular class.

The general form of a import statement for searching a class is →

```
import package1, package2, package3, classname;
```

eg

```
import mypackage.yourpackage.Myclass;
```

In the above example, my package is the name of toplevel or main package, yourpackage is the name of the package which is the inside the package1 and so on we can add the class namely myclass which is defined inside to main package that is mypackage.

Note that the statement must ends with semicolon. The import statement should appear before any class defination in source file We can also use another approach

for importing system defined packages the way is

```
import. packagename.*;
```

eg `import java.awt.*;`

Here package name denotes a single packages or an hierarchy of packages and the star indicates that the compiler should search the entire package hierarchy when it incounters a class name. The major drawback of this approach is that, it is difficult to determine from which package a particular member can come, this is particularly true when a large number of packages are imported and the advantage is that we need not use the long package name repeatedly programs.

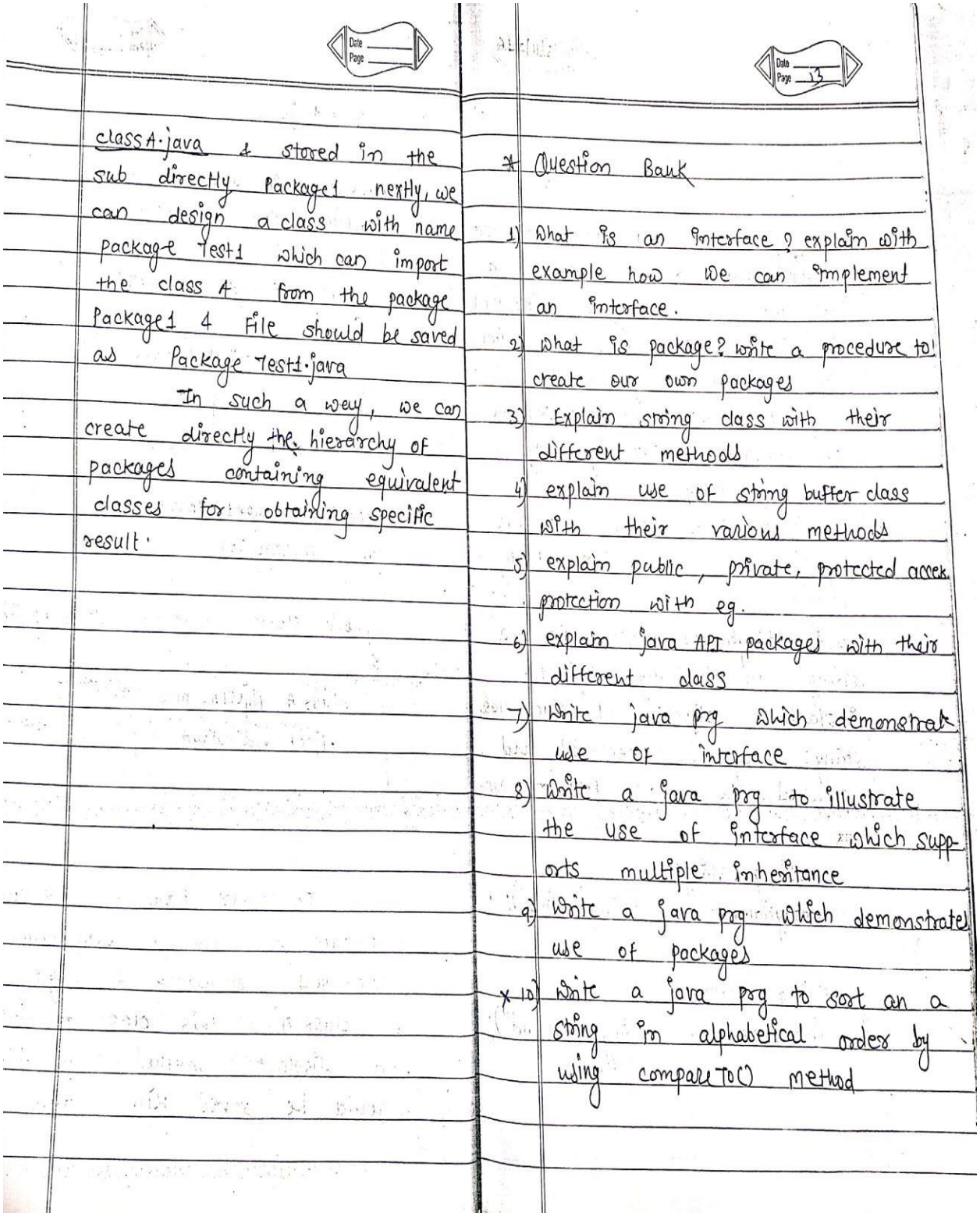
Following program will demonstrate the use of packages in java

class A. java

```
package package1;
public class classA
{
  public void displayA()
  {
    System.out.println("class A");
  }
}
```

```
import package1.classA;
class PackageTest
{
  public static void main(String args[])
  {
    classA objectA= new classA();
    object.displayA();
  }
}
```

In above examples, we can create a package with name 'Package1' containing a single class ie 'class A'. This class A contains an displayA() method. This file should be saved with name



class A.java & stored in the sub directly Package1 nextly, we can design a class with name package Test1 which can import the class A from the package Package1 & File should be saved as Package Test1.java

In such a way, we can create directly the hierarchy of packages containing equivalent classes for obtaining specific result.

* Question Bank

1) What is an Interface ? explain with example how we can implement an Interface.
2) What is package? write a procedure to create our own packages
3) Explain string class with their different methods
4) explain use of string buffer class with their various methods
5) explain public, private, protected access protection with eg.
6) explain java API packages with their different class
7) Write java prg which demonstrate use of interface
8) Write a java prg to illustrate the use of Interface which supports multiple inheritance
9) Write a java prg which demonstrate use of packages
10) Write a java prg to sort an a string in alphabetical order by using compareTo() method

Date Page

* Access protection in java

Like c++ java also uses mainly 3 access protection They are

1) Public
2) Private
3) Protected

:

1) Public access protection -

Any variable or method which can be declared as public has the greatest possible visibility and accessible everywhere within our program.

for eg-

```
public int number;
public void sum()
{
  --
  --
}
```

In the above examples number is an integer type of variable which can be declared with

Date Page 14

public access specifier. sum() is the method which can be declared with public access specifier.

The main use behind using the public access protection is that, if want to declare or use any variable or method of the class, it is possible by simply declaring them as public.

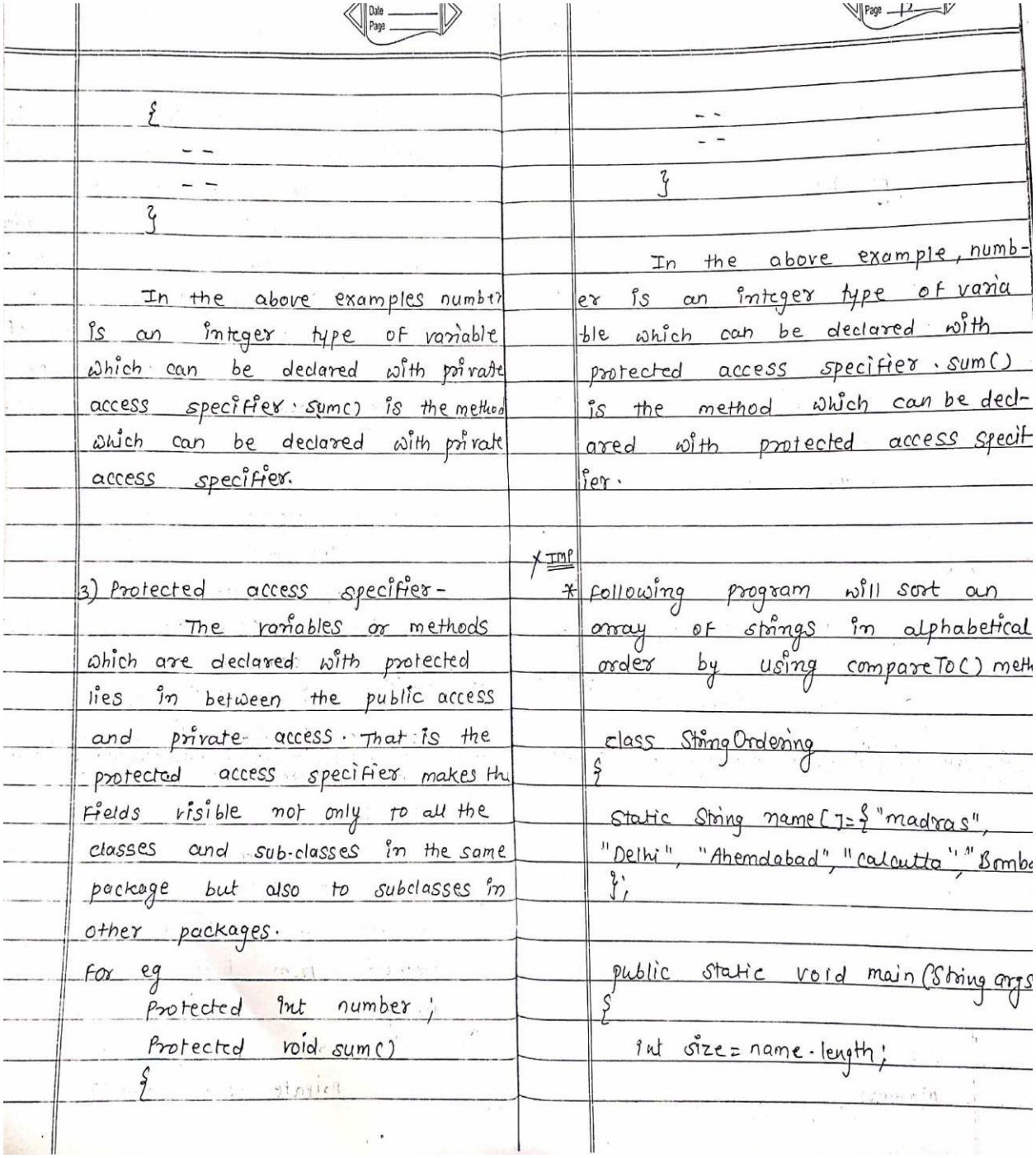
2) Private access protection -

The variables or methods which can be declared with private has highest degree of a protection They are accessible or useful only within their own classes. They cannot be inherited by subclasses and therefore not accessible in a subclasses.

A method which can be declared as private behaves like a method declared as final. It prevents the method from being sub-classed.

for eg -

```
private int number;
private void sum()
```



{
--
--
}

In the above examples number is an integer type of variable which can be declared with private access specifier. sum() is the method which can be declared with private access specifier.

3) Protected access specifier -

The variables or methods which are declared with protected lies in between the public access and private access. That is the protected access specifier makes the fields visible not only to all the classes and sub-classes in the same package but also to subclasses in other packages.

For eg

```
Protected int number;
Protected void sum()
{
--
--
}
```

In the above example, number is an integer type of variable which can be declared with protected access specifier. sum() is the method which can be declared with protected access specifier.

IMP

* following program will sort an array of strings in alphabetical order by using compareTo() meth

```
class StringOrdering
{
  Static String name[]={"madras",
  "Delhi", "Ahemdabad", "calcutta", "Bomb
  };

  public static void main(String args
  {
    int size=name.length;
```

*****