



OBJECT ORIENTED PROGRAMMING WITH JAVA

Dr. N MEHALA

Department of Computer Science and Engineering

Anonymous Inner classes



PES
UNIVERSITY
ONLINE

- *Another category of local inner classes*
- *Classes without any name i.e classes having no name*
- *Can either implements an interface or extends a class.*
- *Can not have more than one instance active at a time.*
- *Whole body of the class is declared in a single statement ending with ;*

Cont...

- Syntax [If extending a class]

```
[variable_type_superclass =] new superclass_name() {  
                                     // properties and methods  
                                     } [;]
```

- Syntax [If implementing an interface]

```
[variable_type_reference =] new reference_name() {  
                                     // properties and methods  
                                     } [;]
```

Anonymous Inner Class Example

```
class A
{
private int a;
A(int a)
{
this.a =a;
}
void show()
{
System.out.println("a="+a);
} // End of show()
} // End of class A
```



```
class innertest1
{
public static void main(String args[])
{
```

Anonymous inner class extending super class A

```
A a1 = new A(20) {
    public void show()
    {
        super.show();
        System.out.println("Hello");
    }
    public void display()
    {
        System.out.println("Hi");
    }
};
```

```
a1.show();
// a1.display();
}
```

Calling show from inner class

```
interface X
{
    int sum(int a,int b);
    int mul(int x,int y);
}

class innertest2
{
    public static void main(String args[])
    {
```

Anonymous inner class implementing an interface

```
X x1 = new X()
{
    public int sum(int a,int b)
    {
        return a+b;
    }
    public int mul(int a,int b)
    {
        return a*b;
    }
};
```

```
System.out.println(x1.sum(10,20));
System.out.println(x1.mul(10,20));
} // End of main
} // End of innertest2
```

Home Exercise

- Write 5 BOX Comparator classes using anonymous inner classes.

Packages

- Classes can be grouped in a collection called *package*
 - *Package names are dot separated, e.g., java.lang.*
 - Java's standard library consists of hierarchical packages, such as java.lang and java.util
<http://java.sun.com/j2se/1.4.2/docs/api>
 - Main reason to use package is to guarantee the uniqueness of class names
 - classes with same names can be encapsulated in different packages
 - tradition of package name: reverse of the company's Internet domain name
e.g. hostname.com -> com.hostname
- i.e. Packages Avoid name space collision. There can not be two classes with same name in a same Package But two packages can have a class with same name.***
- ***Exact Name of the class is identified by its package structure. << Fully Qualified Name>>***

java.lang.String ; java.util.Arrays; java.io.BufferedReader ; java.util.Date

Why do we need packages?

- (1) Higher efficiency, easy to manage.
- (2) Safety. Via “package” modifier, you can define variables only usable within package.
- (3) Each to name classes and variables. Do not have to worry about a name like “Helloworld” has been used by somebody somewhere in the world.

General format

- The general format of a Java source file is as follows:

[package xxx;]

[some import statements;]

a public class;

[some package private classes;]

Java 2 Platform Packages

<u>java.applet</u>	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
<u>java.awt</u>	Contains all of the classes for creating user interfaces and for painting graphics and images.
<u>java.awt.color</u>	Provides classes for color spaces.
<u>java.awt.event</u>	Provides interfaces and classes for dealing with different types of events fired by AWT components.
<u>java.beans</u>	Contains classes related to developing <i>beans</i> -- components based on the JavaBeans™ architecture.
<u>java.beans.beancontext</u>	Provides classes and interfaces relating to bean context.
<u>java.io</u>	Provides for system input and output through data streams, serialization and the file system.
<u>java.lang</u>	Provides classes that are fundamental to the design of the Java programming language.
<u>java.math</u>	Provides classes for performing arbitrary-precision integer arithmetic (BigInteger) and arbitrary-precision decimal arithmetic.
<u>java.net</u>	Provides the classes for implementing networking applications.
<u>java.nio</u>	Defines buffers, which are containers for data, and provides an overview of the other NIO packages.
<u>java.rmi</u>	Provides the RMI package.
<u>java.security</u>	Provides the classes and interfaces for the security framework.
<u>java.sql</u>	Provides the API for accessing and processing data stored in a data source (usually a relational database) using the Java™ programming language.
<u>java.text</u>	Provides classes and interfaces for handling text, dates, numbers, and messages in a manner independent of natural languages.
<u>java.util</u>	Contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes (a string tokenizer, a random-number generator, and a bit array).

Packages provided by Java

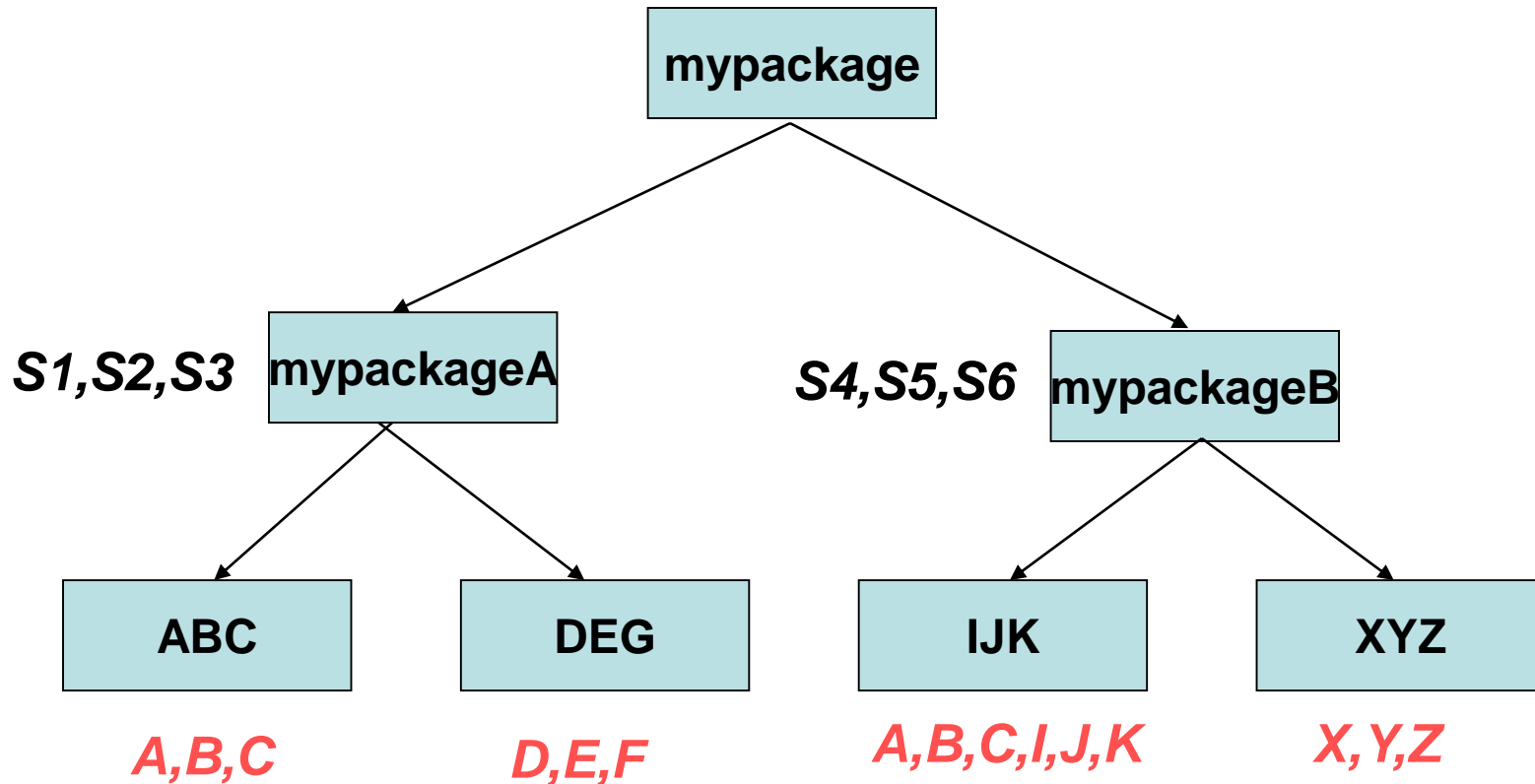
- `java.lang`: includes classes like `Object`, `String`, `System`, `Math`, `Thread` ...
- `java.util`: `Date`, `Hashtable`, `Stack`, `Vector`, ...
- `java.io`: for input/output
- `java.net`: `InetAddress`, `ServerSocket`, `URL`
- `java.awt`: includes classes like `Menu`, `Button`, `Graphics` ...
- `java.applet`: `Applet`, `Audio`, ...

`java.lang` is always implicitly imported.

Define Your Own Package

- To be included in a package A, use this statement.
package A;
// then define your class
- The package structure must match precisely your directory structure.
i.e. starting from classes directory, package A corresponds to sub-directory A, all the classes for package A should be put in directory A. A sub-package B of package A corresponds to a sub-directory B of directory A. And so on.

Exercise Creating Packages



Package ABC and IJK have classes with same name.

A class in ABC has name `mypackage.mypackageA.ABC.A`

A class in IJK has name `mypackage.mypackageB.IJK.A`

Example

```
package rabbit; //declare belong to rabbit package
public class MotherRabbit{
    public void say(){
        System.out.println("MotherRabbit");
    }
}
```

- Create directory rabbit.
- Put this file MotherRabbit.java in directory rabbit
- Compile it.

Add more to this package

```
/*file: SonRabbit.java*/  
package rabbit; //declare belong to rabbit package  
public class SonRabbit {  
    void say() {  
        System.out.println("sonrabbit");  
    }  
}
```

- Also put this file in rabbit directory
- Compile it.

Create a sub-package for rabbit

```
/* Run1.java */  
package rabbit.run; //this is a subpackage of rabbit  
public class Run1{  
    void run() {  
        System.out.println("Run, Hare, Run!");  
    }  
}
```

- Under rabbit directory, create run directory.
- Put Run1.java in run directory.
- Compile it. To import, use:

import rabbit.run.*

Note, import rabbit.* only imports two classes, not the sub-package run, or its classes.

Importing a package's classes

- There are two ways of using classes from a package. (a) Specify the path, use directly like:

```
public class Example extends java.applet.Applet { .. }
```

(b) Use the import statement:

```
import java.applet.Applet; // or java.applet.*  
public class Example extends Applet { ... }
```

- Usually, when using *, it will import ALL classes, affect efficiency. Better just import what you want.

Package Structure and Naming

- Package structure matches the tree-like directory structure.
- Naming: if you are developing some package that is useful on the web. You should give it a good name. The naming scheme could be reversal of your web address plus the name.

Variables & Methods Access Protection



- Here is a table of access rights:

	No modifier	private	private protected	protected	public
=====					
same class	yes	yes	yes	yes	yes

Same package subclass	yes	no	yes	yes	yes

same package non-subclass	yes	no	no	yes	yes

diff. package subclass	no	no	yes	yes	yes

diff. package non-subclass	no	no	no	no	yes
=====					

Class importation (1)

- Two ways of accessing **PUBLIC** classes of another package
 - 1) explicitly give the full package name before the class name.
 - E.g.

```
java.util.Date today = new java.util.Date( );
```
 - 2) import the package by using the `import` statement at the top of your source files (but below package statements). No need to give package name any more.
 - **to import a single class from the java.util package**

```
import java.util.Date;  
Date today = new Date( );
```
 - **to import all the public classes from the java.util package**

```
import java.util.*;  
Date today = new Date( );
```
 - *** is used to import classes at the current package level. It will NOT import classes in a sub-package.**



Sample class:

```
import javax.swing.*;
```

```
public class SampleClass {  
    MenuEvent c;  
}
```

```
%> javac SampleClass.java
```

```
SampleClass.java:4: cannot find symbol
```

```
Symbol   : class MenuEvent
```

```
Location: class SampleClass
```

```
    MenuEvent c;
```

```
    ^
```

```
1 error
```

MenuEvent is a class in the package javax.swing.event, which locates in the package javax.swing. You need this statement:

```
import javax.swing.event.*;
```

Class importation (2)

- What if you have a name conflict?

E.g

```
import java.util.*;
import java.sql.*;
Date today = new Date( );    //ERROR:java.util.Date
                             //or java.sql.Date?
```

- if you only need to refer to one of them, import that class explicitly

```
import java.util.*;
import java.sql.*;
import java.util.Date;
Date today = new Date( );    // java.util.Date
```

- if you need to refer to both of them, you have to use the full package name before the class name

```
import java.util.*;
import java.sql.*;
java.sql.Date today = new java.sql.Date( );
java.util.Date nextDay = new java.util.Date( );
```



See this code:

```
import java.lang.Math;

public class importTest {
    double x = sqrt(1.44);
}
```

Compile:

```
%> javac importTest.java
importTest.java:3: cannot find symbol
symbol   : method sqrt(double)
location: class importTest
double x = sqrt(1.44);
           ^
1 error
```

?

**For the static members, you need to refer them as
className.memberName**

Static importation

- In J2SE 5.0, importation can also be applied on static fields and methods, not just classes. You can directly refer to them after the static importation.

- E.g. import all static fields and methods of the Math class

```
import static java.lang.Math.*;  
double x = PI;
```

- E.g. import a specific field or method

```
import static java.lang.Math.abs;  
double x = abs(-1.0);
```

- Any version before J2SE 5.0 does NOT have this feature!

Encapsulation of classes into a package

- Add a class into a package — two steps:
 1. put the name of the package at the top of your source file

```
package com.hostname.corejava;  
public class Employee {  
    . . .  
}
```

2. put the files in a package into a subdirectory which matches the full package name

→ stored in the file "Employee.java" which is stored under "somePath/com/hostname/corejava/"

To emphasize on data encapsulation (1)



PES
UNIVERSITY
ONLINE

Let's see a sample class first

```
public class Body {
    public long idNum;
    public String name = "<unnamed>";
    public Body orbits = null;
    public static long nextID = 0;

    Body( ) {
        idNum = nextID++;
    }

    Body(String bodyName, Body orbitsAround) {
        this( );
        name = bodyName;
        orbits = orbitsAround;
    }
}
```

Problem: all the fields are exposed to change by everybody

To emphasize on data encapsulation (2)

improvement on the previous sample class with data encapsulation

```
public class Body {  
    private long idNum;  
    private String name = "<unnamed>";  
    private Body orbits = null;  
    private static long nextID = 0;  
  
    Body( ) {  
        idNum = nextID++;  
    }  
  
    Body(String bodyName, Body orbitsAround) {  
        this( );  
        name = bodyName;  
        orbits = orbitsAround;  
    }  
}
```

Problem: but how can you access the fields?

To emphasize on data encapsulation (3)

improvement on the previous sample class with accessor methods

```
public class Body {
    private long idNum;
    private String name = "<unnamed>";
    private Body orbits = null;
    private static long nextID = 0;

    Body( ) {
        idNum = nextID++; }

    Body(String bodyName, Body orbitsAround) {
        this( );
        name = bodyName;
        orbits = orbitsAround; }

    public long getID() {return idNum;}
    public String getName() {return name;};
    public Body getOrbits() {return orbits;}
}
```

Note: now the fields `idNum`, `name` and `orbits` are read-only outside the class. Methods that access internal data are called *accessor methods* sometime



To emphasize on data encapsulation (4)

modification on the previous sample class with methods setting fields

```
class Body {
    private long idNum;
    private String name = "<unnamed>";
    private Body orbits = null;
    private static long nextID = 0;

    // constructors omitted for space problem. . .

    public long getID() {return idNum;}
    public String getName() {return name;};
    public Body getOrbits() {return orbits;}

    public void setName(String newName) {name = newName;}
    public void setOrbits(Body orbitsAround) {orbits =
        orbitsAround;}
}
```

Note: now users can set the `name` and `orbits` fields. But `idNum` is still read-only

- ☞ Making fields private and adding methods to access and set them enables the users adding actions in the future
- ☞ Don't forget the `private` modifier on a data field when necessary! The default access modifier for fields is `package`



THANK YOU

Dr. N MEHALA

Department of Computer Science and Engineering

mehala@pes.edu



OBJECT ORIENTED PROGRAMMING WITH JAVA

Dr. N MEHALA

Department of Computer Science and Engineering

Input tokens

- **Token:** A unit of user input, separated by whitespace.
 - A Scanner splits a file's contents into tokens.
- If an input file contains the following:

23 3.14
"John Smith"

The Scanner can interpret the tokens as the following types:

<u>Token</u>	<u>Type(s)</u>
23	int, double, String
3.14	double, String
"John Smith"	String String



Files and input cursor

- Consider a file numbers.txt that contains this text:

```
308.2
 14.9 7.4 2.8

3.9 4.7  -15.4
 2.8
```

- A Scanner views all input as a stream of characters:

```
308.2\n 14.9 7.4 2.8\n\n3.9 4.7  -15.4\n 2.8\n
```

^

- input cursor:** The current position of the Scanner.

Consuming tokens

- **consuming input:** Reading input and advancing the cursor.
 - Calling `nextDouble` etc. moves the cursor past the current token.

```
308.2\n    14.9 7.4    2.8\n\n3.9  
4.7    -  
15.4\n    2.8\n
```

```
double x = input.nextDouble();  
// 308.2
```

```
308.2\n    14.9 7.4    2.8\n\n3.9  
4.7    -  
15.4\n    2.8\n
```

File input question

- Recall the input file `numbers.txt`:

```
308.2
    14.9  7.4   2.8
  3.9  4.7      -15.4
        2.8
```

- Write a program that reads the first 5 values from the file and prints them along with their sum.

```
number = 308.2
number = 14.9
number = 7.4
number = 2.8
number = 3.9
Sum = 337.2
```

File input answer

```
// Displays the first 5 numbers in the given file,  
// and displays their sum at the end.
```

```
import java.io.*;    // for File  
import java.util.*;  // for Scanner  
  
public class Echo {  
    public static void main(String[] args)  
        throws FileNotFoundException {  
        Scanner input = new Scanner(new  
        File("numbers.txt"));  
        double sum = 0.0;  
        for (int i = 1; i <= 5; i++) {  
            double next = input.nextDouble();  
            System.out.println("number = " + next);  
            sum = sum + next;  
        }  
        System.out.printf("Sum = %.1f\n", sum);  
    }  
}
```

Scanner exceptions

- `InputMismatchException`
 - You read the wrong type of token (e.g. read "hi" as `int`).
- `NoSuchElementException`
 - You read past the end of the input.

Reading an entire file

- Suppose we want our program to process the entire file.
(It should work no matter how many values are in the file.)

```
number = 308.2  
number = 14.9  
number = 7.4  
number = 2.8  
number = 3.9  
number = 4.7  
number = -15.4  
number = 2.8  
Sum = 329.3
```

Testing for valid input

- `Scanner` methods to see what the next token will be:

Method	Description
<code>hasNext()</code>	returns <code>true</code> if there are any more tokens of input to read <i>(always true for console input)</i>
<code>hasNextInt()</code>	returns <code>true</code> if there is a next token and it can be read as an <code>int</code>
<code>hasNextDouble()</code>	returns <code>true</code> if there is a next token and it can be read as a <code>double</code>

- These methods do not consume input; they just give information about the next token.
 - Useful to see what input is coming, and to avoid crashes.

File input question 2

- Modify the `Echo` program to process the entire file:
(It should work no matter how many values are in the file.)

```
number = 308.2  
number = 14.9  
number = 7.4  
number = 2.8  
number = 3.9  
number = 4.7  
number = -15.4  
number = 2.8  
Sum = 329.3
```

File input answer 2

```
// Displays each number in the given file,  
// and displays their sum at the end.  
import java.io.*;      // for File  
import java.util.*;    // for Scanner  
public class Echo {  
    public static void main(String[] args)  
        throws FileNotFoundException {  
        Scanner input = new Scanner(new  
File("numbers.txt"));  
        double sum = 0.0;  
        while (input.hasNextDouble()) {  
            double next = input.nextDouble();  
            System.out.println("number = " + next);  
            sum = sum + next;  
        }  
        System.out.printf("Sum = %.1f\n", sum);  
    }  
}
```

File input question 3

- Modify the `Echo` program to handle files that contain non-numeric tokens (by skipping them).
- For example, it should produce the same output as before when given this input file, `numbers2.txt`:

```
308.2  hello
      14.9 7.4  bad stuff    2.8

3.9 4.7  oops  -15.4
:-)    2.8  @#*($&
```



File input answer 3

```
// Displays each number in the given file,  
// and displays their sum at the end.  
import java.io.*;          // for File  
import java.util.*;        // for Scanner  
public class Echo2 {  
    public static void main(String[] args)  
        throws FileNotFoundException {  
        Scanner input = new Scanner(new File("numbers2.txt"));  
        double sum = 0.0;  
        while (input.hasNext()) {  
            if (input.hasNextDouble()) {  
                double next = input.nextDouble();  
                System.out.println("number = " + next);  
                sum = sum + next;  
            } else {  
                input.next();    // throw away the bad token  
            }  
        }  
        System.out.printf("Sum = %.1f\n", sum);  
    }  
}
```

Weather question

- Write a program that reads in temperatures and outputs the coldest and warmest temps.

Weather.txt file data

16.2 23.5

19.1 7.4

22.8

18.5 -1.8 14.9

PARTIAL SOLUTION

```
public static void main(String[] args) throws FileNotFoundException{
    Scanner input = new Scanner(new File("weather.txt"));
    double max=0;
    double min=0;
    while(input.hasNextDouble()){
        double current = input.nextDouble();
        if (current < min){
            min = current;
        }
        if (current > max){
            max = current;
        }
    }
    System.out.println("The coldest temp in data set was " + min);
    System.out.println("The warmest temp in data set was " + max);
}
```

Writing to a File

- We will use a `PrintWriter` object to write to a file

Writing to a File

- The out field of the System class is a `PrintWriter` object associated with the console
 - We will associate our `PrintWriter` with a file now

```
PrintWriter fout = new PrintWriter("output.txt");  
fout.println(29.95);
```

```
fout.println("Hello, World!");
```

- This will print the exact same information as with `System.out` (except to a file “output.txt”)!

Closing a File

- Only main difference is that we have to close the file stream when we are done writing
- If we do not, not all output will be written
- At the end of output, call `close()`

```
fout.close();
```

Closing a File

- Why?
 - When you call `print()` and/or `println()`, the output is actually written to a buffer. When you close or flush the output, the buffer is written to the file
 - The slowest part of the computer is hard drive operations – much more efficient to write once instead of writing repeated times

File Locations

- When determining a file name, the default is to place in the same directory as your .class files
- If we want to define other place, use an absolute path (e.g. c:\My Documents)

```
in = new  
    FileReader("c:\\homework\\input.dat");
```

Sample Program

- Two things to notice:
 - Have to import from java.io
 - I/O requires us to catch checked exceptions
 - `java.io.IOException`

Java Output Review

- CONSOLE:

```
System.out.print("To the screen");
```

- FILE:

```
PrintWriter fout =  
    new PrintWriter(new File("output.txt"));  
fout.print("To a file");
```



THANK YOU

Dr. N MEHALA

Department of Computer Science and Engineering

mehala@pes.edu

Naming conventions

- **Package names:** start with lowercase letter
 - E.g. java.util, java.net, java.io . . .
- **Class names:** start with uppercase letter
 - E.g. File, Math . . .
 - avoid name conflicts with packages
 - avoid name conflicts with standard keywords in java system
- **Variable, field and method names:** start with lowercase letter
 - E.g. x, out, abs . . .
- **Constant names:** all uppercase letters
 - E.g. PI . . .
- **Multi-word names:** capitalize the first letter of each word after the first one
 - E.g. helloWorldApp, getName . . .
- **Exception class names:** (1) start with uppercase letter (2) end with “Exception” with normal exception and “Error” with fatal exception
 - E.g. OutOfMemoryError, FileNotFoundException

Accessor and Mutator Methods



Accessor Methods

- *Accessor methods are used to return the values of instance fields.*
- *Accessor methods do not receive any parameter*
- *Accessor methods are defined according to the type of the instance fields [One Accessor method for each instance field]*
- *Accessor methods should have public scope*
- *General Form of declaring Accessor Method:*

<return type> get<InstanceField>()

{

return InstanceField;

}

Insert get word before field name

<return type> should be the type of instance field

Example

```
class Student
{
private String name;
private String idno;
private int age;
private double amount;
```

**Define Accessor Methods for
All the instance fields**

```
// accessor method for name
public String getName()
{
return name;
}
// accessor method for idno
public String getIdno()
{
return name;
}
// accessor method for age
public int getAge()
{
return name;
}
// accessor method for amount
public double getAmount()
{
return name;
} .....
} // End of class Student
```

Mutator Methods

- *Mutator methods are used to set the values of instance fields.*
- *Mutator methods have return type as void and receives the parameter of type of instance field*
- *Mutator methods are defined according to the type of the instance fields [One mutator method for each instance field]*
- *Mutator methods should have public scope*
- *General Form of declaring Mutator Method:*

Type should be type of instance field

```
void set<InstanceField>(<type> parameterName )  
{  
  
}
```

Insert set word before field name

<return type> should be void

Example



PES
UNIVERSITY
ONLINE

```
class Student  
{  
private String name;  
private String idno;  
private int age;  
private double amount;
```

Define Mutator Methods for
All the instance fields

```
// mutator method for name  
public void setName(String name)  
{  
    this.name = name;  
}  
// mutator method for idno  
public void setIdno(String idno)  
{  
    this.idno = idno;  
}  
// mutator method for age  
public void setAge(int age)  
{  
    this.age = age;  
}  
// mutator method for amount  
public void setAmount(double amount)  
{  
    This.amount = amount;  
    ..... }  
} // End of class Student
```

Exercise

An instructor of a class wants to record following items of information for each student of a class of 80 students:

- 1. Name of student*
- 2. Id no of each student*
- 3. Three test marks for three subjects namely “OOP” ,”Data Structures” and “Software Engineering” [Use 2-Dimensional array 3 * 3 to hold the values. Row index indicates subjects and column index indicates test values]*

Supply the following operations :

- 1. Accessor methods for name and idno fields*
- 2. Accessor method for retrieving/getting a particular test value for given subject and test*
- 3. Accessor method for retrieving/getting all test values for given subject*
- 4. Accessor method for retrieving/getting all subject scores for a given test_no*



Exercise cont....

5. Supply the mutator operations for all the four mentioned cases
6. Supply a method for getting student with highest for a given subject
7. Supply a method for getting student with highest for a given test_no

<i>test index_no</i>		0	1	2
<i>Subject index</i>	<i>OOP</i> → 0	Test1 OOP	Test 2 OOP	Test 3 OOP
	<i>DS</i> → 1			
	<i>SE</i> → 2			

marks[][]

Example cont..

```
public class Student  
{  
private String name;  
private String idno;  
private double[][] marks;
```

Instance Fields

```
Student(String name, String idno)  
{  
this.idno = idno;  
this.name = name;  
marks = new double[3][3];  
}
```

Constructor



// Accessor Methods for name and idno

```
public String getName()  
{  
    return name;  
}  
  
public String getIdno()  
{  
    return idno;  
}
```

// Mutator method for name and idno

```
public void setName(String name)  
{  
    this.name = name;  
}  
  
public void setIdno(String idno)  
{  
    this.idno = idno;  
}
```




// Accessor method for getting marks for a given subject and testno
public double getTestmarks(int subject,int test_no)

{
return marks[subject-1][test_no-1];
}

// Accessor method for getting all test marks for a given subject
public double[] getSubjectMarks(int subject)

{
double scores[] = new double[3];
scores[0] = marks[subject-1][0];
scores[1] = marks[subject-1][1];
scores[2] = marks[subject-1][2];
return scores;
}



// Accessor method for getting all subject marks for a given test

```
public double[] getTestMarksForAllSubjects(int test_no)  
{  
double scores[] = new double[3];  
scores[0] = marks[0][test_no-1];  
scores[1] = marks[1][test_no-1];  
scores[2] = marks[2][test_no-1];  
return scores;  
}
```

// Accessor Method for all test scores

```
public double[][] getAllTestMarks()  
{  
return marks;  
}
```



// Mutator method for setting marks for a given subject and testno
public void setTestmarks(int subject,int test_no, double value)

{
marks[subject-1][test_no-1] = value;
}

// Mutator method for setting all test marks for a given subject

public void setSubjectMarks(int subject, double[] scores)

{
marks[subject-1][0] = scores[0] ;
marks[subject-1][1] = scores[1] ;
marks[subject-1][2] = scores[2] ;
}



// Mutator method for setting all subject marks for a given test

```
public void setTestMarksForAllSubjects(int test_no, double[] scores)  
{  
marks[0][test_no-1] = scores[0];  
marks[1][test_no-1] = scores[1];  
marks[2][test_no-1] = scores[2];  
}
```

// Mutator Method for all test scores

```
public void setAllTestMarks(double[][] scores)  
{  
marks = scores;  
}
```

Rest Methods Take Home Exercise

Immutable class

- ***Class whose object's state can not be changed after creation is immutable.***
- ***State of Object -→ Instance field Values***
- ***If an object's attribute's values can not be changed after creation then that object is also immutable***
- ***Example : String class***
- ***Alternatively if an object's state can be changed after creation then that object is mutable object and class is mutable class e.g StringBuffer***

How to make a class immutable

- *Define instance fields as private and do not supply any mutator method*

```
class Box
{
    private double length;
    private double length;
    private double length;
    Box(double l, double b, double h)
    {
        ...
    }
    double area()
    {
        .....
    }
} // End of Box classs
```

***Box class is
Immutable***

What's the Advatage of Immutable class

***Their Object references
can be freely shared among
many clients without any risk***

Example 2 Immutable class

```
class Triangle  
{  
  // Instance fields  
  private double a,b,c;  
  // Constructor Methods  
  Triangle(double a)  
  {  
  }  
  Triangle(double a,double b)  
  {  
  }  
  Triangle(double a,double b,double c)  
  {  
  }  
  // Accessor Method for side1  
  double getA()  
  {  
  }  
  // Accessor Method for side2  
  double getB()  
  {  
  }  
  // Accessor Method for side3  
  double getC()  
  {  
  }  
  // Method for computing area  
  double area()  
  {  
  }  
  // Method for computing perimeter  
  double perimeter()  
  {  
  }  
} // End of Triangle class
```



THANK YOU

Dr. N MEHALA

Department of Computer Science and Engineering

mehala@pes.edu