Essentials of Information Technology

PC-CS-305

Constructors

Objectives

In this lecture, we will

- Introduce constructors
- Discuss overloading constructors
- Review encapsulation

gauravgambhir.cse@piet.co.in

Constructors

- A constructor creates an Object of the class and by initializing all the instance variables and creating a place in memory to hold the Object.
- Constructor is use to create instance of a class.
- If you don't define any constructor, the default constructor defined by Object class is used to create the instance. ***Demo.java***
- In case, you want to initialize your object at the time of creation, you have to define constructor. ***NameDemo.java***
- Hence, without constructor, you can't create any object in java.

Constructors

- A constructor is called to instantiate the class when new is used
  - An object is constructed
  - The code in the constructor initialises the member variables
- A constructor is like method with the same name as the class
- A constructor has no return type
- For example:

```
class MyClass
{
     int n;
     MyClass ( ) { n = 0; }              MyClass.java
     …
```

Constructors

```
class MyClass
{
 int n;
 MyClass ( )
 {
    n = 0;
 }
 MyClass (int num)
 {
    n = num;
 }
 void MyClass()
 {
        System.out.println("Value of n is " + n);
 }
 public static void main(String args[])
 {
        MyClass m1=new MyClass();
        m1.MyClass();
        MyClass m2=new MyClass(10);
        m2.MyClass();
 }
}
```

```
---------- Run ----------
Value of n is 0
Value of n is 10

Output completed (0 sec consumed) - Normal Termination
```

hir.cse@piet.co.in

Constructor vs. Method

- Constructor and methods differ in three aspects
  - return type , name, modifiers.
- Methods have valid ***return type*** or no return type i.e. void , Constructors have no return type , not even void.
- Constructors have same ***name*** as that of class, methods use names other than class names.
- Methods can have any ***modifier*** like abstract , final, static etc. ,but constructors cannot be abstract , final , static.
- Example ***Test.java***

gauravgambhir.cse@piet.co.in

Constructors and Arguments

- A constructor that takes no arguments is referred to as a default constructor
- A constructor can take one or more arguments
- For example:

  MyClass (int num) { n = num; }
- Constructors can be overloaded
  - More than one constructor can be provided
  - The signatures of the constructors must be different
  - A signature refers to the number and types of the arguments

Example ***MyClass.java , Test.java***

gauravgambhir.cse@piet.co.in

Constructors and Arguments

```
class Test
{
    int i,j,sum;
    Test()   // try appending final
    {
        i=10;
        j=20;
    }
    Test(int p,int q)
    {
        i=p;
        j=q;
    }
    final void Test()
    {
        sum=i+j;
        System.out.println(sum);
    }
    public static void main(String[] args)
    {
        Test t1 = new Test();
        Test t2 = new Test(100,200);
        t1.Test();
        t2.Test();

    }
}
```

```
---------- Run ----------
30
300

Output completed (0 sec consumed) - Normal Termination
```

gauravgambhir.cse@piet.co.in

Overloading of constructors

- When constructors having same name but different arguments are provided in same class , the constructors are said to be overloaded.

```
public class Person {

  String name;
  int age;

  /*public Person (String theName)
  {
    name = theName; age = 0;
  }*/
  public Person (String theName)
  {
    this(theName,0);
  }
  public Person (String theName, int theAge)
  {
    name = theName; age = theAge;
  }
```

Example ***Person.java***

gauravgambhir.cse@piet.co.in

Overloaded Constructors

```
public static void main(String args[])
{
  Person fred=new Person("Fred Bloggs");
  Person bill=new Person("Bill Bailey",62);
  //Person thePerson=new Person(); will give compile time error
  int billsAge=bill.getAge();
  System.out.println(billsAge);

  /*int fredAge=fred.getAge();
  System.out.println(fredAge);*/
}

/*  int getAge()
  {
    return this.age;
  }
*/
    int getAge()
    {
      return this.age;
    }

    void setAge(int age)
    {
      this.age=age;
    }
}
```

```
---------- Run ----------
62

Output completed (0 sec consumed) - Normal Termination
```

@piet.co.in

Constructors

- What happens if an attempt is made to build a Person object without supplying any arguments?
  - Person thePerson = new Person ( );
- This does not match either of the two constructors given on the previous slide
  - Java refuses to build an object
- If NO constructors are provided by the programmer then Java will provide a “do-nothing” constructor
- If at least one constructor is written by the programmer Java no longer provides the “do-nothing” constructor
  - It is good practice to write one yourself!

gauravgambhir.cse@piet.co.in

The this Keyword

- Within instance method or constructor, this is a reference to the current object --- the object whose method or constructor is being called.
- You can refer to any member of current object from within instance method or a constructor by using this.

```
public class Point {
        public int x=0;
        public int y=0;
        public Point(int a , int b){
            x=a;
            y=b;
        }
    }
```

```
public class Point {
        public int x=0;
        public int y=0;
        public Point(int x , int y){
            this.x=x;
            this.y=y;
        }
}
```

Point.java

gauravgambhir.cse@piet.co.in

Using this in Constructor

- From within a constructor you can also use this to call another constructor in same class.

```
public class Rectangle{
          private int x, y;
          private int width, height;

          public Rectangle(){
              this(0,0,0,0);
           }

          public Rectangle(int width, int height){
               this(0,0,width,height);
           }

           public Rectangle(int x, int y , int width, int height){
                this.x=x;
                this.y=y;
                this.width=width;
                this.height=height;
            }
```

Rectangle.java

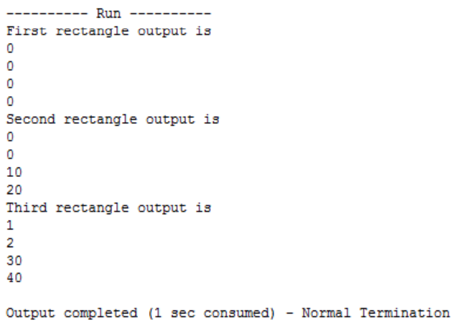
gauravgambhir.cse@piet.co.in

Using this in Constructor

```
            public static void main(String args[]){
                Rectangle r1=new Rectangle();
                System.out.println("First rectangle output is");
                System.out.println(r1.x);
                System.out.println(r1.y);
                System.out.println(r1.width);
                System.out.println(r1.height);

                Rectangle r2=new Rectangle(10,20);
                System.out.println("Second rectangle output is");
                System.out.println(r2.x);
                System.out.println(r2.y);
                System.out.println(r2.width);
                System.out.println(r2.height);

                Rectangle r3=new Rectangle(1,2,30,40);
                System.out.println("Third rectangle output is");
                System.out.println(r3.x);
                System.out.println(r3.y);
                System.out.println(r3.width);
                System.out.println(r3.height);
            }
}
```

gauravgambhir.cse@piet.co.in

Using this in Constructor



```
---------- Run ----------
First rectangle output is
0
0
0
0
Second rectangle output is
0
0
10
20
Third rectangle output is
1
2
30
40

Output completed (1 sec consumed) - Normal Termination
```

gauravgambhir.cse@piet.co.in

Using this in Constructors

- The variable this can be used in constructors to avoid repetition of code:

  public Person (String theName, int theAge)

    {name = theName; age = theAge;}

  public Person (String theName)

    { this(theName, 0);}

- In this example the second constructor makes use of the first constructor
  - this(theName,0) asks the compiler to use the constructor in this object that takes two arguments; a string and an integer

Example ***Person.java*** ***this_demo.java***

gauravgambhir.cse@piet.co.in

Using this in Constructors

```
public class this_Demo {

     private int day = 1;
     private int month = 1;
     private int year = 2000;

     public this_Demo(int day, int month, int year) {
        this.day   = day;
        this.month = month;
        this.year  = year;
     }

     public this_Demo(){
         this(10,20,30);
     }

     void print(){
         System.out.println("day  "+day);
         System.out.println("month  "+month);
         System.out.println("year  "+year);
     }
     public static void main(String args[]){
         this_Demo d=new this_Demo();
         d.print();
         this_Demo d1=new this_Demo(1,2,3);
         //d1.print();
     }

}
```

@piet.co.in

Encapsulation

- One objective of OO is to encourage code re-use
- If a class is to be re-used then it is essential that users of the class do not depend on or make assumptions about the internals of the class
- OO languages encourage the use of classes
- In order to use a class:
  - it is necessary to know what the class can do, the functionality
  - but NOT how it does it, the implementation
- Encapsulation is concerned with exposing the functionality but hiding the implementation
- Access modifiers helps in achieving encapsulation in Java.

gauravgambhir.cse@piet.co.in

Access Modifiers

Members of a Java class can be:

- public
  - Accessible by users of the class
- private
  - Hidden, not available, to users of the class
  - Accessible within the class by other methods of the class.
- default
  - It is declared by not writing any access modifier at all.
  - Default access level means code inside the class itself + code inside classes in the same package as this class, can access the class , field , constructor or method.
  - Sometimes also called package access modifier.

gauravgambhir.cse@piet.co.in

Access Modifiers

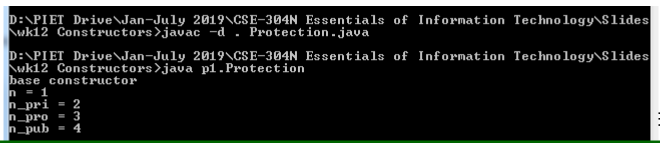
- protected
  - protected access modifier does the same as the default access, except subclasses can also access protected methods and member variables of the super class.
  - This is true even if the subclass is not located in same package as the super class.

- Example : Examples/P1, Examples/P2, Examples/P3

gauravgambhir.cse@piet.co.in

Access Modifiers

```java
package p1;

public class Protection {

int n = 1;
private int n_pri = 2;
protected int n_pro = 3;
public int n_pub = 4;


public Protection()  {

    System.out.println("base constructor");
    System.out.println("n = " + n);
    System.out.println("n_pri = " + n_pri);
    System.out.println("n_pro = " + n_pro);
    System.out.println("n_pub = " + n_pub);
}
public static void main(String args[]){
    Protection p=new Protection();
}
}
```

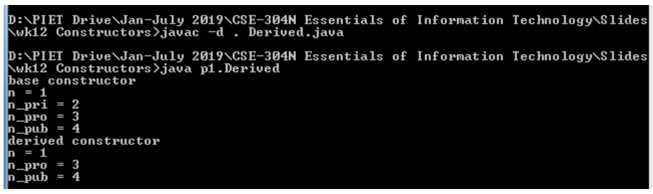


```
D:\PIET Drive\Jan-July 2019\CSE-304N Essentials of Information Technology\Slides
\wk12 Constructors>javac -d . Protection.java

D:\PIET Drive\Jan-July 2019\CSE-304N Essentials of Information Technology\Slides
\wk12 Constructors>java p1.Protection
base constructor
n = 1
n_pri = 2
n_pro = 3
n_pub = 4
```

et.co.in

Access Modifiers

```java
package p1;

class Derived extends Protection {

    Derived()  {
        System.out.println("derived constructor");

        System.out.println("n = " + n);
        // class only
         //System.out.println("n_pri = " + n_pri);
        System.out.println("n_pro = " + n_pro);
        System.out.println("n_pub = " + n_pub);
    }

    public static void main(String args[]){
        Derived d=new Derived();
    }
}
```

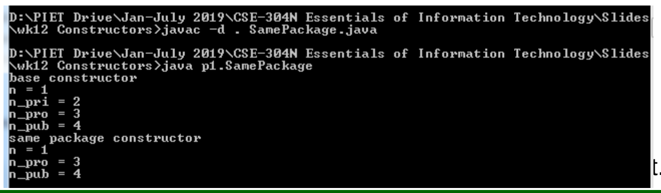


```
D:\PIET Drive\Jan-July 2019\CSE-304N Essentials of Information Technology\Slides
\wk12 Constructors>javac -d . Derived.java

D:\PIET Drive\Jan-July 2019\CSE-304N Essentials of Information Technology\Slides
\wk12 Constructors>java p1.Derived
base constructor
n = 1
n_pri = 2
n_pro = 3
n_pub = 4
derived constructor
n = 1
n_pro = 3
n_pub = 4
```

iet.co.in

Access Modifiers

```
package p1;

class SamePackage {

SamePackage() {
     Protection p = new Protection();
     System.out.println("same package constructor");
     System.out.println("n = " + p.n);
     // class only
     //System.out.println("n_pri = " + p.n_pri);
     System.out.println("n_pro = " + p.n_pro);
     System.out.println("n_pub = " + p.n_pub);
}

public static void main(String args[]){
     SamePackage p=new SamePackage();
}
}
```

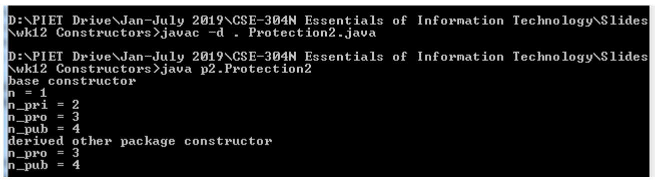


```
D:\PIET Drive\Jan-July 2019\CSE-304N Essentials of Information Technology\Slides
\wk12 Constructors>javac -d . SamePackage.java

D:\PIET Drive\Jan-July 2019\CSE-304N Essentials of Information Technology\Slides
\wk12 Constructors>java p1.SamePackage
base constructor
n = 1
n_pri = 2
n_pro = 3
n_pub = 4
same package constructor
n = 1
n_pro = 3
n_pub = 4
```

t.co.in

Access Modifiers

```
package p2;
public class Protection2 extends p1.Protection {
public Protection2() {
     System.out.println("derived other package constructor");
     // class or package only
     //System.out.println("n = " + n);
     // class only
     // System.out.println("n_pri = " + n_pri);
     System.out.println("n_pro = " + n_pro);
     System.out.println("n_pub = " + n_pub);
     }
     public static void main(String args[]){
          Protection2 p2=new Protection2();
     }
}
```

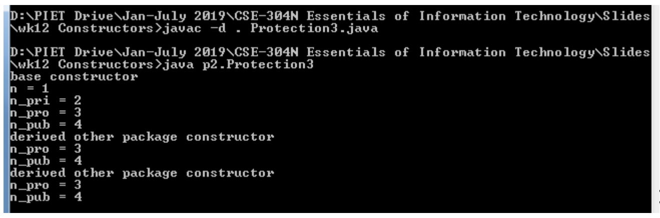


```
D:\PIET Drive\Jan-July 2019\CSE-304N Essentials of Information Technology\Slides
\wk12 Constructors>javac -d . Protection2.java

D:\PIET Drive\Jan-July 2019\CSE-304N Essentials of Information Technology\Slides
\wk12 Constructors>java p2.Protection2
base constructor
n = 1
n_pri = 2
n_pro = 3
n_pub = 4
derived other package constructor
n_pro = 3
n_pub = 4
```

gauravgambhir.cse@piet.co.in

Access Modifiers

```java
package p2;
 class Protection3 extends Protection2 {
    Protection3() {
        System.out.println("derived other package constructor");
    // class or package only
    // System.out.println("n = " + n);
    // class only
    // System.out.println("n_pri = " + n_pri);
    System.out.println("n_pro = " + n_pro);
    System.out.println("n_pub = " + n_pub);
}
    public static void main(String args[]){
        //p1.Protection p = new p1.Protection();
        //System.out.println(p.n_pro);
        Protection3 p3=new Protection3();

    }
}
```

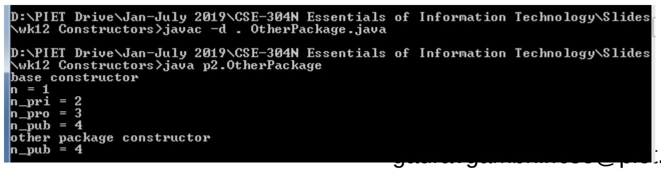


```
D:\PIET Drive\Jan-July 2019\CSE-304N Essentials of Information Technology\Slides
\wk12 Constructors>javac -d . Protection3.java

D:\PIET Drive\Jan-July 2019\CSE-304N Essentials of Information Technology\Slides
\wk12 Constructors>java p2.Protection3
base constructor
n = 1
n_pri = 2
n_pro = 3
n_pub = 4
derived other package constructor
n_pro = 3
n_pub = 4
derived other package constructor
n_pro = 3
n_pub = 4
```

iet.co.in

Access Modifiers

```java
package p2;

class OtherPackage {
    OtherPackage() {
    p1.Protection p = new p1.Protection();
    System.out.println("other package constructor");
    // class or package only
    // System.out.println("n = " + p.n);
    // class only
    // System.out.println("n_pri = " + p.n_pri);
    // class, subclass or package only
    // System.out.println("n_pro = " + p.n_pro);
    System.out.println("n_pub = " + p.n_pub);
    }
    public static void main(String args[]){
    OtherPackage op=new OtherPackage();
    }
}
```

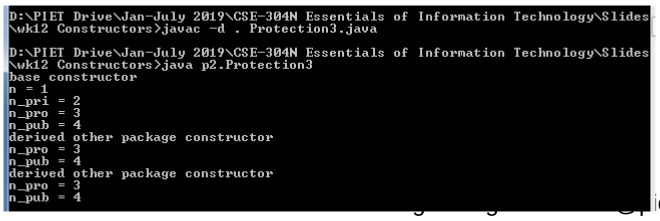


```
D:\PIET Drive\Jan-July 2019\CSE-304N Essentials of Information Technology\Slides
\wk12 Constructors>javac -d . OtherPackage.java

D:\PIET Drive\Jan-July 2019\CSE-304N Essentials of Information Technology\Slides
\wk12 Constructors>java p2.OtherPackage
base constructor
n = 1
n_pri = 2
n_pro = 3
n_pub = 4
other package constructor
n_pub = 4
```

.co.in

Access Modifiers

```
package p2;
 class Protection3 extends Protection2 {
    Protection3() {
        System.out.println("derived other package constructor");
    // class or package only
    // System.out.println("n = " + n);
    // class only
    // System.out.println("n_pri = " + n_pri);
    System.out.println("n_pro = " + n_pro);
    System.out.println("n_pub = " + n_pub);
}
    public static void main(String args[]){
        //p1.Protection p = new p1.Protection();
        //System.out.println(p.n_pro);
        Protection3 p3=new Protection3();

    }
}
```

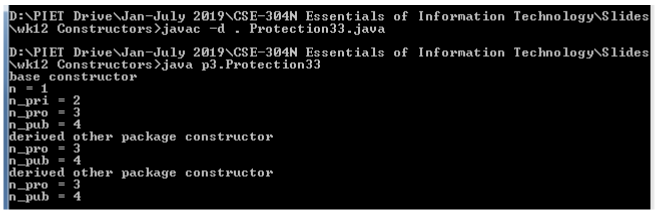


```
D:\PIET Drive\Jan-July 2019\CSE-304N Essentials of Information Technology\Slides
\wk12 Constructors>javac -d . Protection3.java

D:\PIET Drive\Jan-July 2019\CSE-304N Essentials of Information Technology\Slides
\wk12 Constructors>java p2.Protection3
base constructor
n = 1
n_pri = 2
n_pro = 3
n_pub = 4
derived other package constructor
n_pro = 3
n_pub = 4
derived other package constructor
n_pro = 3
n_pub = 4
```

iet.co.in

Access Modifiers

```
package p3;
class Protection33 extends p2.Protection2 {
    Protection33() {
        System.out.println("derived other package constructor");
        // class or package only
        // System.out.println("n = " + n);
        // class only
        // System.out.println("n_pri = " + n_pri);
        System.out.println("n_pro = " + n_pro);
        System.out.println("n_pub = " + n_pub);
    }
    public static void main(String args[]){
        Protection33 p3=new Protection33();
    }
}
```



```
D:\PIET Drive\Jan-July 2019\CSE-304N Essentials of Information Technology\Slides
\wk12 Constructors>javac -d . Protection33.java

D:\PIET Drive\Jan-July 2019\CSE-304N Essentials of Information Technology\Slides
\wk12 Constructors>java p3.Protection33
base constructor
n = 1
n_pri = 2
n_pro = 3
n_pub = 4
derived other package constructor
n_pro = 3
n_pub = 4
derived other package constructor
n_pro = 3
n_pub = 4
```

gauravgambhir.cse@piet.co.in

Access Modifiers

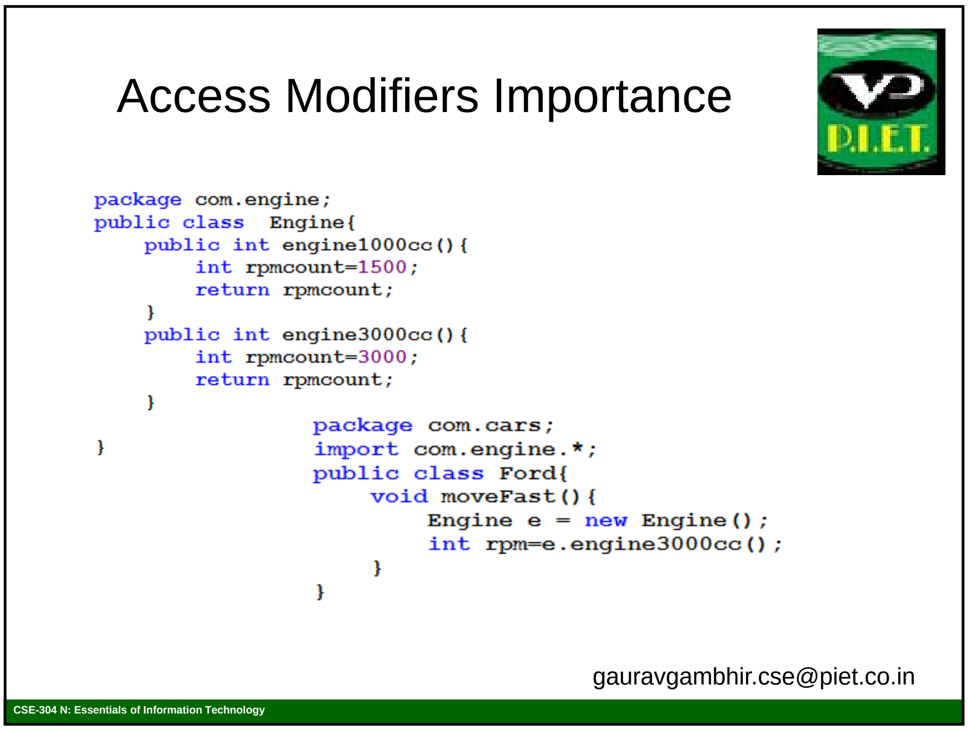
Access Levels				
Visibility	public	protected	default	private
From same class	Y	Y	Y	Y
From class in same package	Y	Y	Y	N
From a class from outside package	Y	N	N	N
From a sub class outside the package	Y	Y(through inheritance)	N	N

gauravgambhir.cse@piet.co.in

Access Modifiers Importance

- Access modifiers helps to implement the concept of encapsulation in OOP.
- Encapsulation helps to hide the data and behaviour.
- Access modifiers affect the accessibility at two levels
  - Class
  - Members (methods and instance variables)
- Access modifier at class level
  - **public** When a class is marked as public , it is visible to all other classes which are inside and outside the package. i.e. visible to everyone
  - **Example :** Engine.java , Ford.java
  - A class marked as public is accessible to all classes in other packages, but should mention the import statement.
  - There is no need for an import statement when a class inside same package uses the class marked as public.
  - **default** it has no keyword, when there is no access modifier declared , the access is default access.

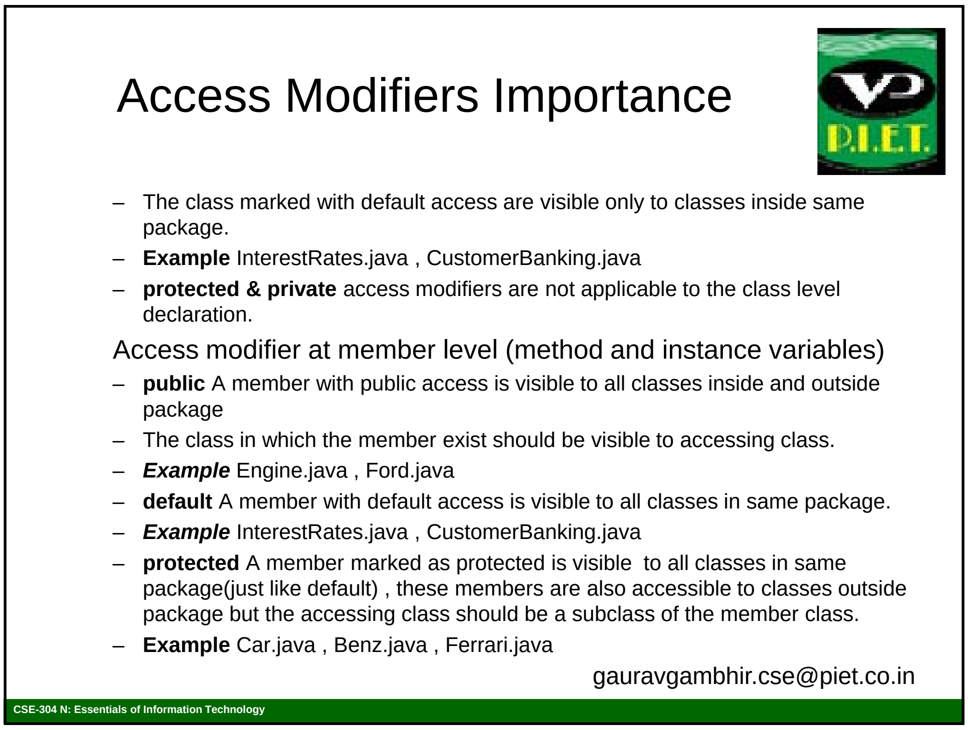
gauravgambhir.cse@piet.co.in



Access Modifiers Importance

```
package com.engine;
public class  Engine{
    public int engine1000cc(){
        int rpmcount=1500;
        return rpmcount;
    }
    public int engine3000cc(){
        int rpmcount=3000;
        return rpmcount;
    }
                  package com.cars;
}                 import com.engine.*;
                  public class Ford{
                      void moveFast(){
                          Engine e = new Engine();
                          int rpm=e.engine3000cc();
                      }
                  }
```

gauravgambhir.cse@piet.co.in



Access Modifiers Importance

- The class marked with default access are visible only to classes inside same package.
- **Example** InterestRates.java , CustomerBanking.java
- **protected & private** access modifiers are not applicable to the class level declaration.

Access modifier at member level (method and instance variables)

- **public** A member with public access is visible to all classes inside and outside package
- The class in which the member exist should be visible to accessing class.
- ***Example*** Engine.java , Ford.java
- **default** A member with default access is visible to all classes in same package.
- ***Example*** InterestRates.java , CustomerBanking.java
- **protected** A member marked as protected is visible to all classes in same package(just like default) , these members are also accessible to classes outside package but the accessing class should be a subclass of the member class.
- **Example** Car.java , Benz.java , Ferrari.java

gauravgambhir.cse@piet.co.in

Access Modifiers Importance

```
package com.bank;
class InterestRates {

    double creditCardInterest(){
        return 10.11 ;
    }

    double homeLoanInterest(){
        return 10.11;
    }
}
```

```
package com.Customerbank;
public class CustomerBanking
{
    double calculateHomeLoan(){
        InterestRates i = new InterestRates();
        double interest=i.creditCardInterest();
        return interest;
    }

}
```

gauravgambhir.cse@piet.co.in

Access Modifiers Importance

```
package com.cars;
public class Car{
    protected int speed;
}
```

```
package com.cars;
public class Benz{
    int move(){
        Car c = new Car();
        return c.speed;
    }
}
```

```
package com.fastCars;
import com.cars.Car;
public class Ferrari extends Car{
    int moveFast(){
        return super.speed; // speed is accessible this way
    }
    int move(){
        Car c = new Car();
        c.speed=10;  // speed is not accessible using an instance
    }
}
```

gauravgambhir.cse@piet.co.in

Access Modifiers Importance

- **private** When a member is marked as private, it is only visible to other members inside same class.
- Other classes inside and outside the package will not be able to access the private members of the class.
- **Example** Computer.java , Thief.java

- **Why class cannot be protected or private**
- When we say protected for a member we are making a contract at the class level that means the protected member is accessible to the classes that are outside the package only through inheritance, so we are making a contract at class level, now if a class is marked as protected then obviously there is no higher level to make a contract that means it doesn’t have any meaningful message , so protected is not legal access modifier for a class.
- When we mark a class as private any other class will not be able to view that class or able to access it, in that case we cannot create particular instance of class and that class is of no use.

gauravgambhir.cse@piet.co.in

Access Modifiers Importance

```
public class Computer{
    private String keyCode;
}


  public class Thief{
      void steal(){
          Computer c= new Computer();
          System.out.println(c.keyCode);//not accessible
      }
  }
```

gauravgambhir.cse@piet.co.in

Code Sample

```
public class Person {
    private String name;
    private int age;
    public Person (String theName)
    {name = theName; age = 0;}
     public Person (String theName, int theAge)
    {name = theName; age = theAge;}

    public void setName (String theName) {name = theName;}
     public void setAge (int theAge) {age = theAge;}
     public String getName () {return name;}
     public int getAge () {return age;}
```

gauravgambhir.cse@piet.co.in

Code Sample (continued)

```
     public void displayDetails ()
    {
       printHeader();
      System.out.println("Name is : " + name );
       System.out.println("Age is  : " + age );
    }
    private void printHeader()
    {
        System.out.println("********************************");
       System.out.println("Person details");
       System.out.println("********************************");
    }
}
```

Example : Person.java

gauravgambhir.cse@piet.co.in

Access to a Person Object

- Consider the following code:

  Person fred = new Person (“Fred Bloggs”, 63);
- Any attempt to access the public members is allowed:

  fred.displayDetails();

  int n = fred.getAge();
- Any attempt to access private members outside the class is prohibited:

```
fred.age = 64;         // not allowed
                       // the member variable age is private
fred.printHeader();    // not allowed
                       // the member method is private
```

gauravgambhir.cse@piet.co.in

Scope

- In developing classes the scope of variables must be considered
- A member variable, an instance variable, has class scope
  - That is the variable is available from the point it is declared up until the end of the class
  - Any methods of the class can access the variable
- A variable declared within a member method is referred to as a local variable and has method scope
  - That is it is available for use throughout the method but NOT outside the method
  - Example : ScopeTest.java

gauravgambhir.cse@piet.co.in

Scope

```
class ScopeTest
{
    int i; // member variable or instance variable
    public static void main(String[] args)
    {
        ScopeTest t = new ScopeTest();
        t.testScope();
        t.localScope();
    }
    void localScope(){
      int j=0;  // local variable ,need to be initialized before use
      System.out.println(j);
    }
    void testScope(){
        //System.out.println(j);
        System.out.println(i);
    }
}
```

```
---------- Run ----------
0
0

Output completed (0 sec consumed) - Normal Termination
```

gauravgambhir.cse@piet.co.in

Instance Variables

- Instance variables are member variables declared within a class

  public class Person {
      private String name;
      private int age;

  - name and age are instance variables
- They are referred to as instance variables because one instance of the variable is created for each instance of the class
  - Each object has a memory location to store its value of the data described by the member variable

gauravgambhir.cse@piet.co.in

Static Modifier

- static means associated with class rather than particular instance of class.
- static modifier can be applied to method & variables.
- static can work as an independent block also.

Example : Ecstatic.java ,StaticDemo.java , Shirt.java , Jacket.java , TShirt.java

gauravgambhir.cse@piet.co.in

Static Modifier

```
class Ecstatic {

    static int x=0;

        Ecstatic()
        {
            x++;
            System.out.println(x);
        }

    public static void main(String args[])
    {

        //int reallyImportantVariable;

        Ecstatic e1=new Ecstatic();
        Ecstatic e2=new Ecstatic();
        e1.x=100;
        e2.x=200;
        //Ecstatic.x=300;
        //reallyImportantVariable=e1.x;
        //System.out.println(reallyImportantVariable);
        System.out.println(e1.x);
    }
}
```

```
---------- Run ----------
1
2
200

Output completed (0 sec consumed) - Normal Termination
```

gauravgambhir.cse@piet.co.in

Static Modifier

```
class StaticDemo
{
  static int i=10;
  public static void main(String args[])
  {
    abc();
    System.out.println(i);
  }
  static void abc()
  {
    System.out.println("Good Evening");
  }
  static {
    System.out.println("Good Morning");
  }
}
```

```
---------- Run ----------
Good Morning
Good Evening
10

Output completed (0 sec consumed) - Normal Termination
```

gauravgambhir.cse@piet.co.in

Static Modifier

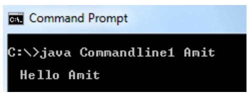
```
public class Shirt{
  public static int quantity=1000;
  //rest of the code
}


public class Jacket extends Shirt{

    public static void main(String args[]){
        System.out.println(Shirt.quantity); //prints out 1000
        Shirt.quantity=Shirt.quantity-100; //Now quantity is 900
    }

}

public class TShirt extends Shirt
{
  public static void main(String args[]){
    System.out.println(Shirt.quantity); //prints 900 because they are referring to the same thing
  }
}
```

```
---------- Run ----------
1000

Output completed (0 sec consumed) - Normal Termination
```

gauravgambhir.cse@piet.co.in

Command Line Arguments

- The command line argument is the argument passed to a program at the time when you run it.
- They are stored as string in **String** array passed to the args parameter of main() method.
- public static void main(String args[])



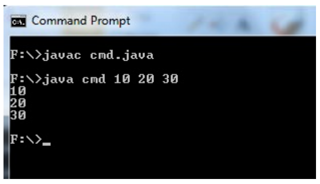
```
Command Prompt
C:\>java Commandline1 Amit
 Hello Amit
```

```
class Commandline1
{
    public static void main(String[] args)
    {
        System.out.println(" Hello " + args[0]);
    }
}
```

gauravgambhir.cse@piet.co.in

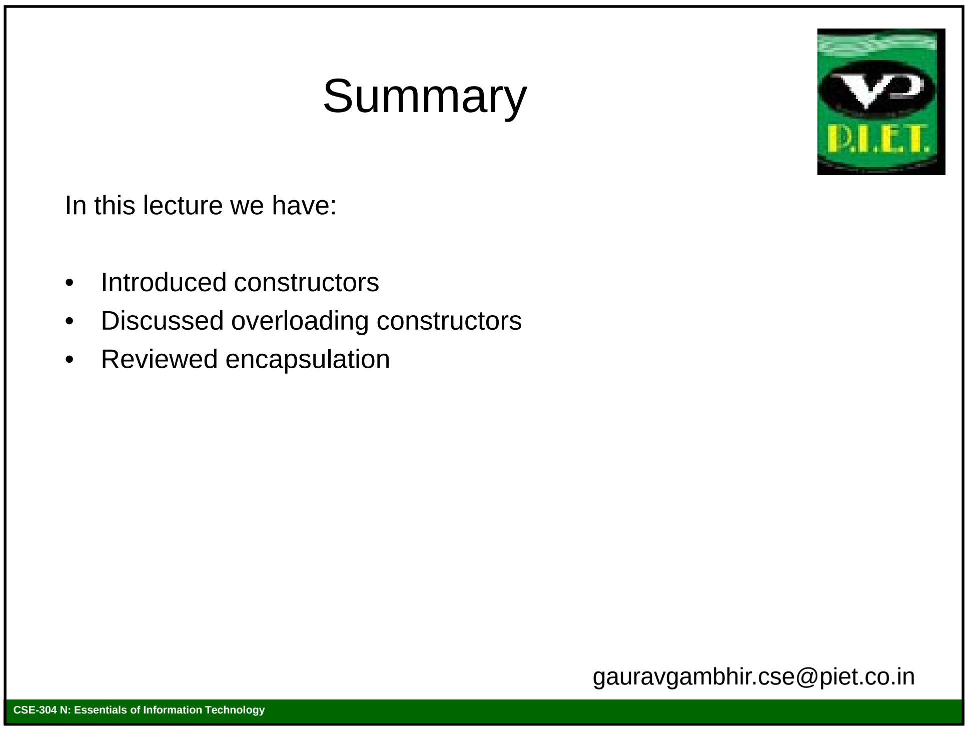
Command Line Arguments

```
class cmd
{
  public static void main(String[] args)
  {
    for(int i=0;i< args.length;i++)
    {
    System.out.println(args[i]);
    }
  }
}
```



```
Command Prompt
F:\>javac cmd.java

F:\>java cmd 10 20 30
10
20
30

F:\>_
```

gauravgambhir.cse@piet.co.in



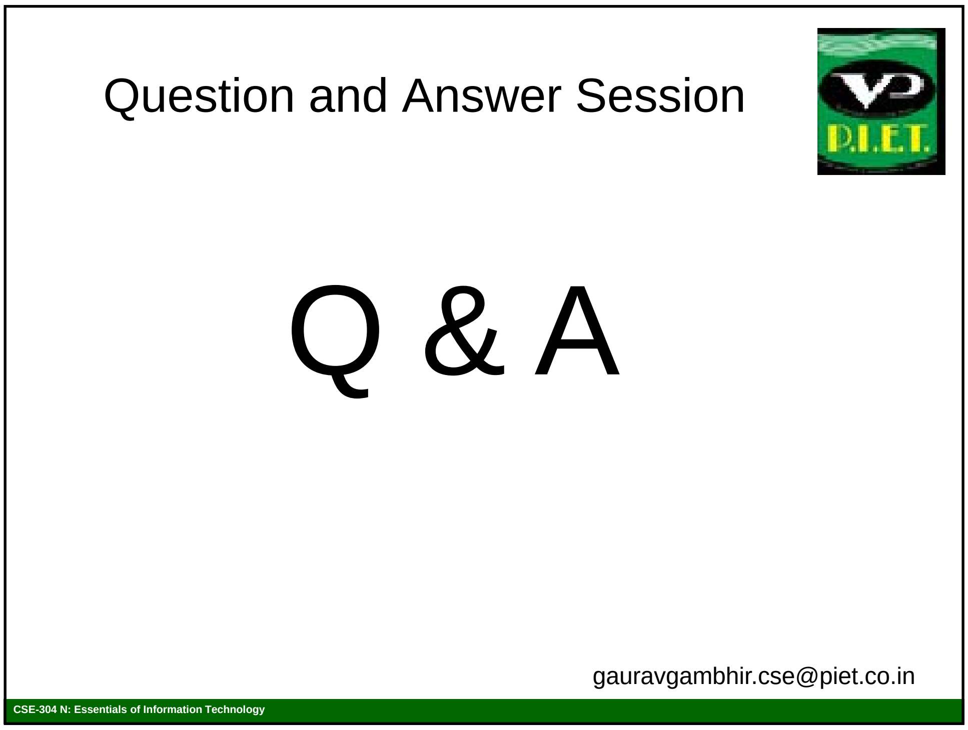
Summary

In this lecture we have:

- Introduced constructors
- Discussed overloading constructors
- Reviewed encapsulation

gauravgambhir.cse@piet.co.in

CSE-304 N: Essentials of Information Technology



Question and Answer Session

Q & A

gauravgambhir.cse@piet.co.in

CSE-304 N: Essentials of Information Technology