****

**IQRA NATIONAL UNIVERSITY HAYATABAD PESHAWAR**

*NAME:* ***MAZAHIR ABBAS***

*ID NO #* ***16746***

*SUBJECT:* ***OOPs***

***PAPER: OOPs***

*SUBMITTED TO:* ***SIR AYYUB (PRORAMMING TEACHER)***

**Write down the following questions each question carries equal marks. Please answer briefly.**

**Q1. a. Why access modifiers are used in java, explain in detail Private and Default Access modifiers?**

**ANSWER:**

* **WHY ACCESS MODIFIERS ARE USED IN JAVA:**

The use of modifiers goes to the core concepts of encapsulation, aka 'data hiding' in object-oriented development.

Variables should never be public. That is the whole point of private/protected modifiers on them: to prevent direct access to the variables themselves.

You provide methods to manipulate variables. A method has to be public in order to allow other programmers (or classes) access to that data. But by using methods, you can control how a variable is manipulated. Which is the entire point because you've hidden the details of the variable behind the method. This is the fundamental concept of OO encapsulation.

The protected modifier only allows for access to the variable and/or method by subclasses and other classes in a package. Typically, protected is used to prevent others from access the data except in controlled ways. Kind of like an "invited guest" status versus anyone on the street just wandering by.

Private is the most restrictive. Only the class itself can access private values. This is useful where you want to prevent subclasses from modifying variables except through the controls provided and deny that ability to everyone else. This is the ultimate encapsulation. Private methods are used to provide internal functionality that you never need or want to expose to anyone else. Very common in framework implementations.

Default is not commonly used. Generally, public is the rule for methods you want others to use. Private is the default for class-level variables and for variables you don't want subclasses to directly access. Protected is typically used within related classes to allow internal access to data but keep it away from outside users. Protected methods are used for internal class behaviors that are needed to be shared or leveraged within a class or family of classes but not exposed for anyone else to see. Private methods are used for pure internal behaviors to a class that will never be shared outside itself, not even to its children.

* **PRIVATE ACCESS MODIFIER:**

Allows a variable or method to only be accessed in the class in which it was created. No other class beyond the class that created the variable or method can access it. This is closely similar to your internal organs. They are only accessible to the owner. To make a variable or method private, you simply append the private keyword before the variable or method type. Let us use private in a coding example. If a bank wants to provide an interest rate of 10% on its loans, it would make sure that the interest rate variable (let us suppose int int\_rate;) would stay private so as no other class would try to access it and change it. ***For example;***

**Private String name;**

The above example creates a variable called name and ensures that it is only accessible within the class from which it was created.

Another example for a method is

**private void setAge(){**

**System.out.println("Set Age");**

**}**

The above example ensures that the method stage is accessible only within the class from which it was created and nowhere else.

* **DEFAULT ACCESS MODIFIER:**

The default access modifier is different from all the other access modifiers in that it has no keyword. To use the default access modifier, you simply use none of the other access modifiers and that simply means you are using a default access modifier.

**For example**, to use the default access modifier for a class, you use

**class Bird{**

**}**

This basically means you are using the default access modifier. The default access modifier allows a variable, method, or class to be accessible by other classes within the same package. A package is a collection of related classes in a file directory. For more information about packages, check out the section on packages.

Any variable, method, or class declared to use the default access modifier cannot be accessed by any other class outside of the package from which it was declared.

**int age;**

**void setNewAge(){**

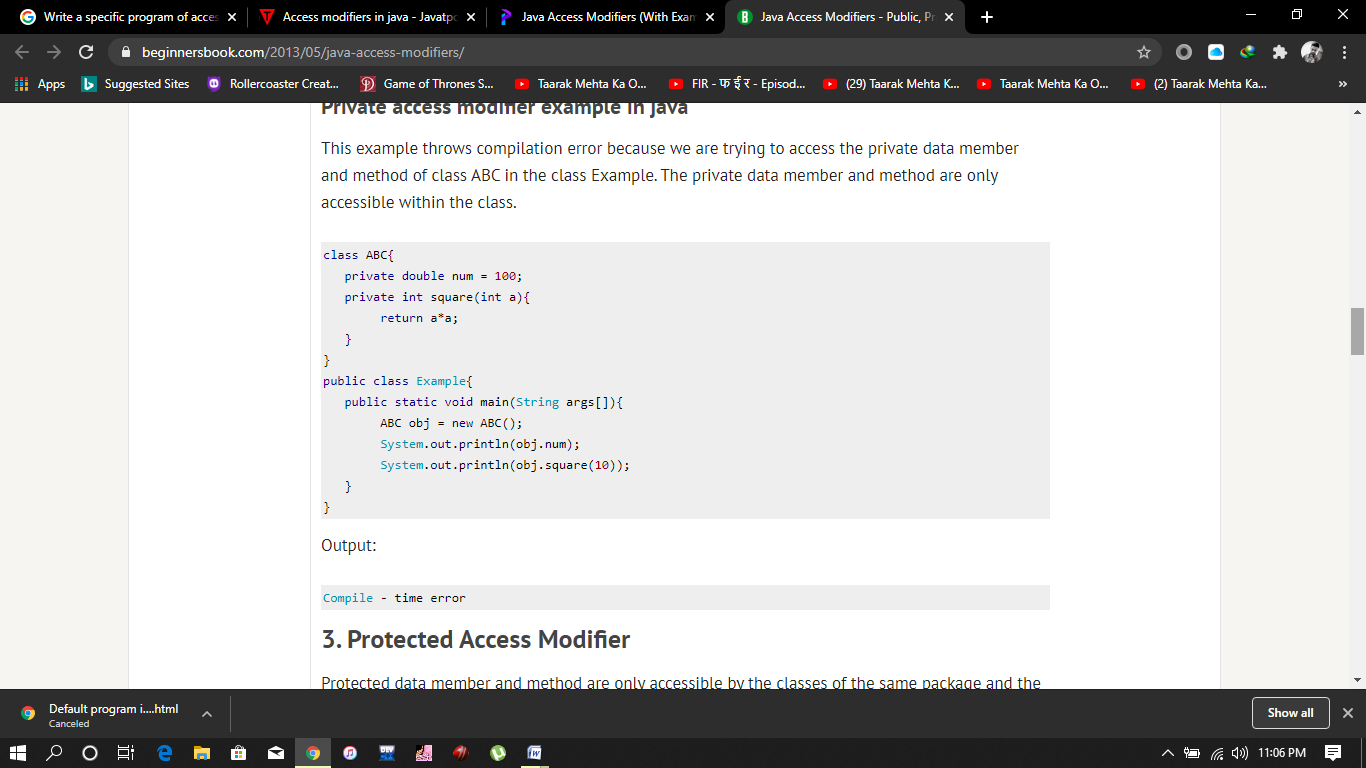
**}**

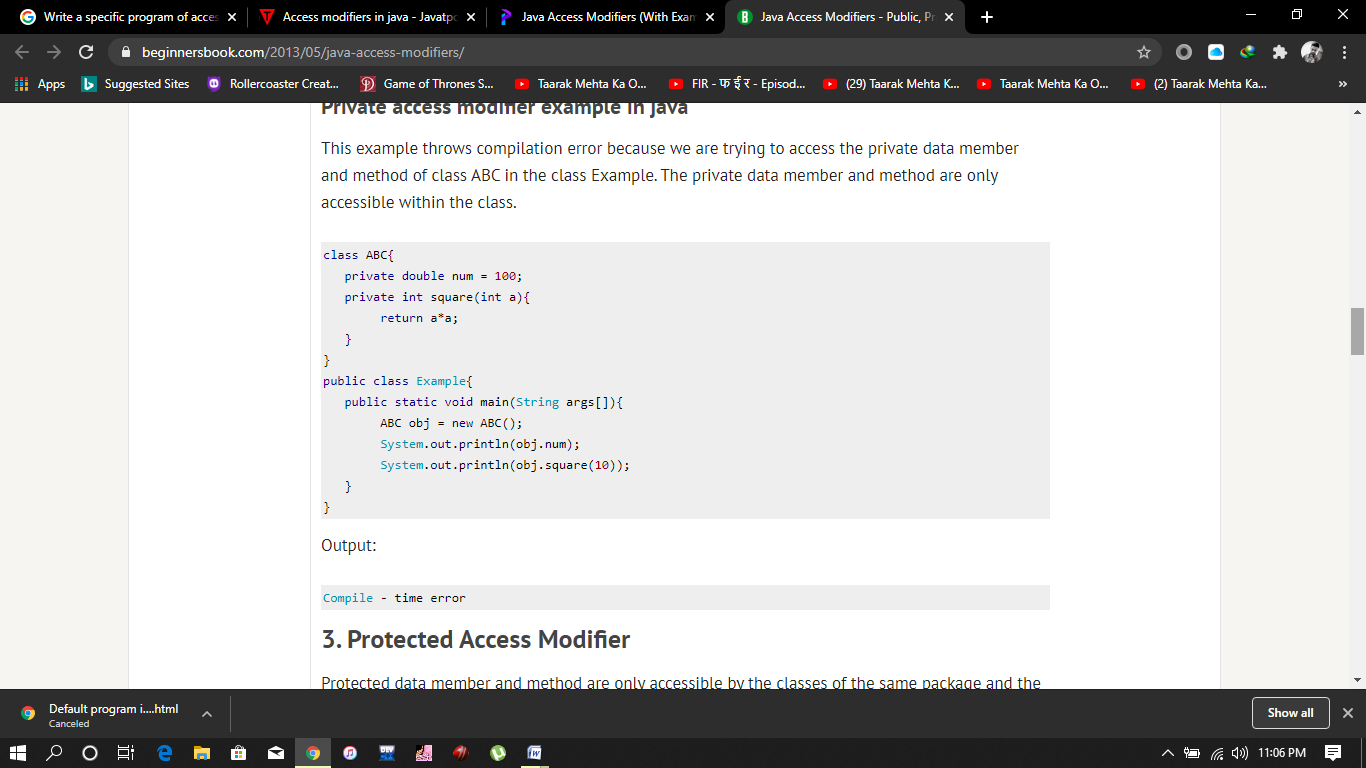
Above are some ways of using the default access modifier for a variable or method. Don’t forget, the default access modifier does not have a key word. The absence of the 3 other access modifiers means you are using the default access modifier.

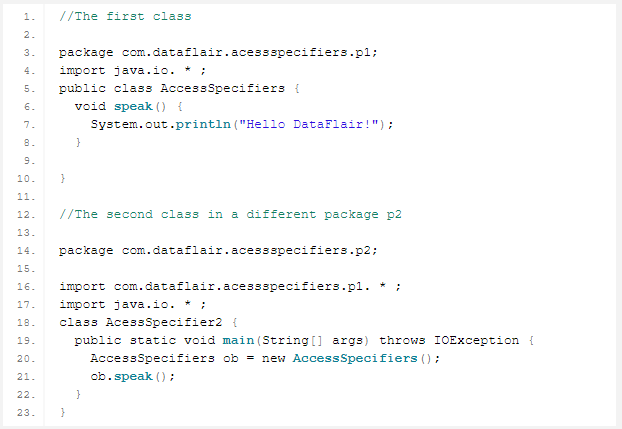
**b. Write a specific program of the above mentioned access modifiers in java.**

**ANSWER:**

* **Program in Private Access Modifier:**



**OUTPUT:**

* **Program in Default Access Modifier:**

****

**OUTPUT:**

**Q2. a. Explain in detail Public and Protected access modifiers?**

**ANSWER:**

* **PUBLIC ACCESS MODIFIER:**

A class, method, constructor, interface, etc. declared public can be accessed from any other class. Therefore, fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.

However, if the public class we are trying to access is in a different package, then the public class still needs to be imported. Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.

**Example**

The following function uses public access control

**public static void main(String[] arguments) {**

**// ...**

**}**

The main () method of an application has to be public. Otherwise, it could not be called by a Java interpreter (such as java) to run the class.

* **PROTECTED ACCESS MODIFIER:**

Variables, methods, and constructors, which are declared protected in a super class, can be accessed only by the subclasses in other package or any class within the package of the protected members' class.

The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in an interface cannot be declared protected.

Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

**Example**

The following parent class uses protected access control, to allow its child class override *openSpeaker()* method −

**class AudioPlayer {**

**protected boolean openSpeaker(Speaker sp) {**

**// implementation details**

**}**

**}**

**class StreamingAudioPlayer extends AudioPlayer {**

**boolean openSpeaker(Speaker sp) {**

**// implementation details**

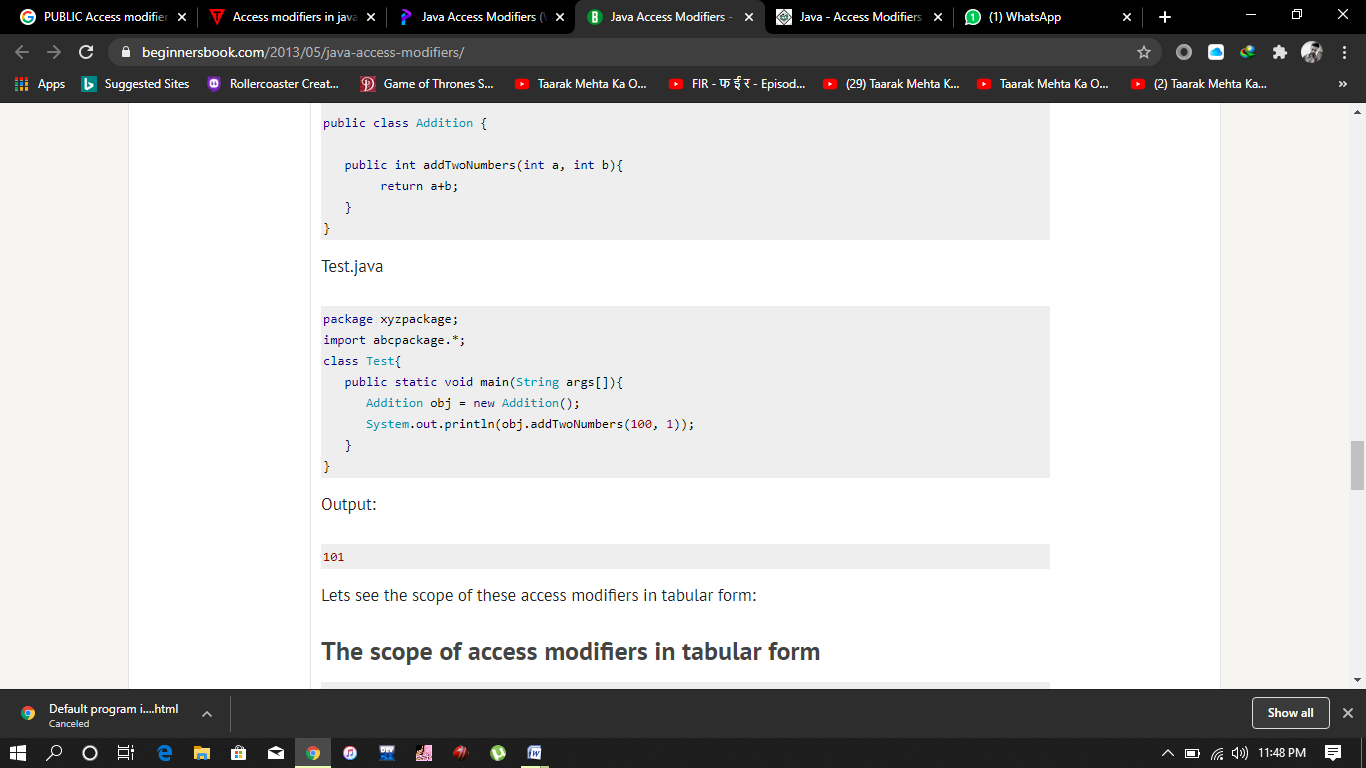
**}**

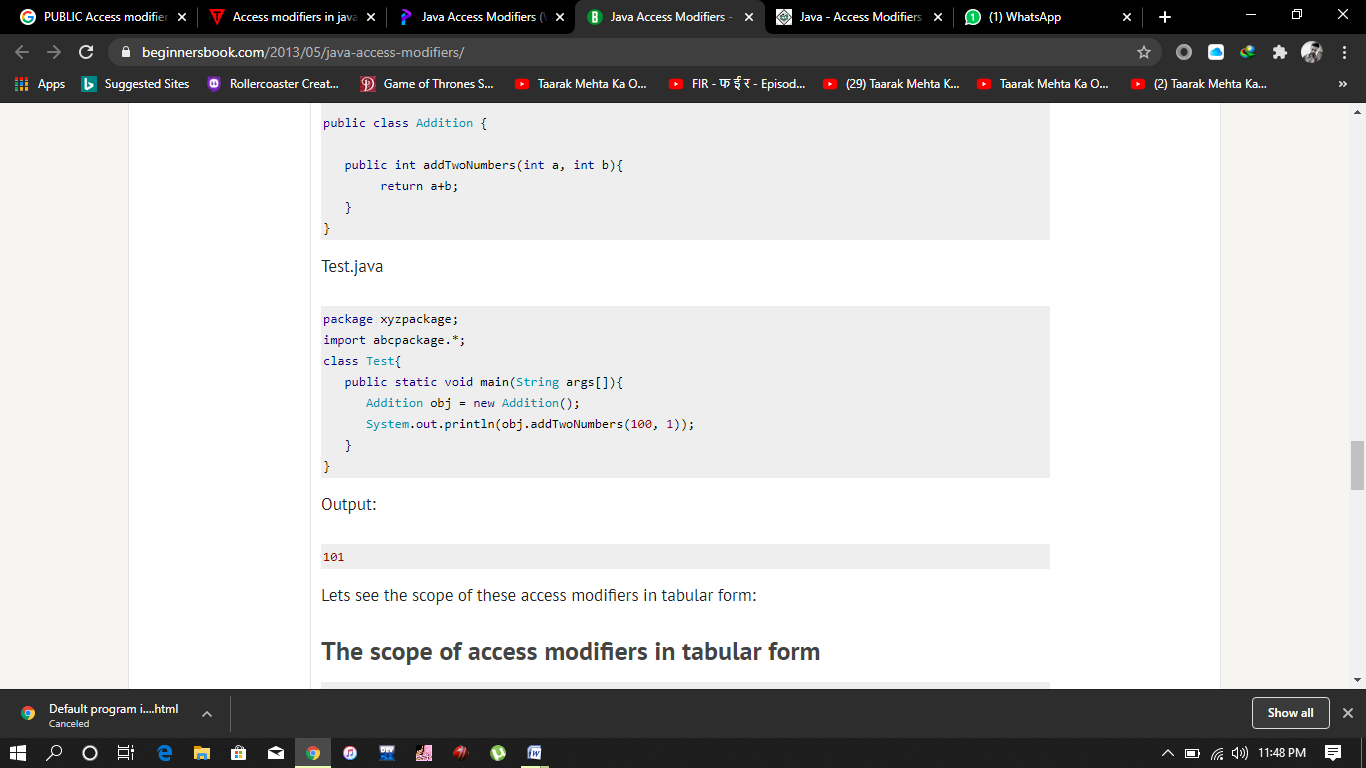
**}**

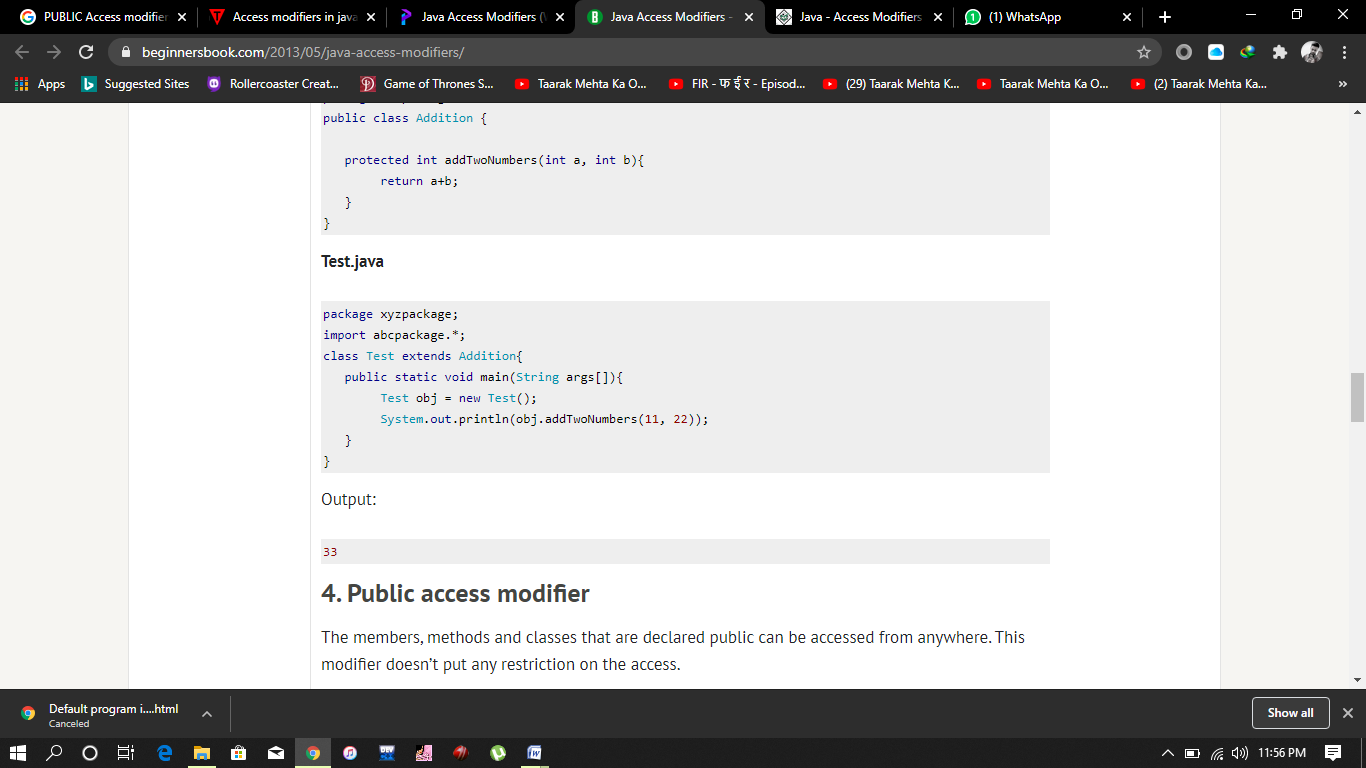
Here, if we define openSpeaker() method as private, then it would not be accessible from any other class other than *AudioPlayer*. If we define it as public, then it would become accessible to all the outside world. But our intention is to expose this method to its subclass only, that’s why we have used protected modifier.

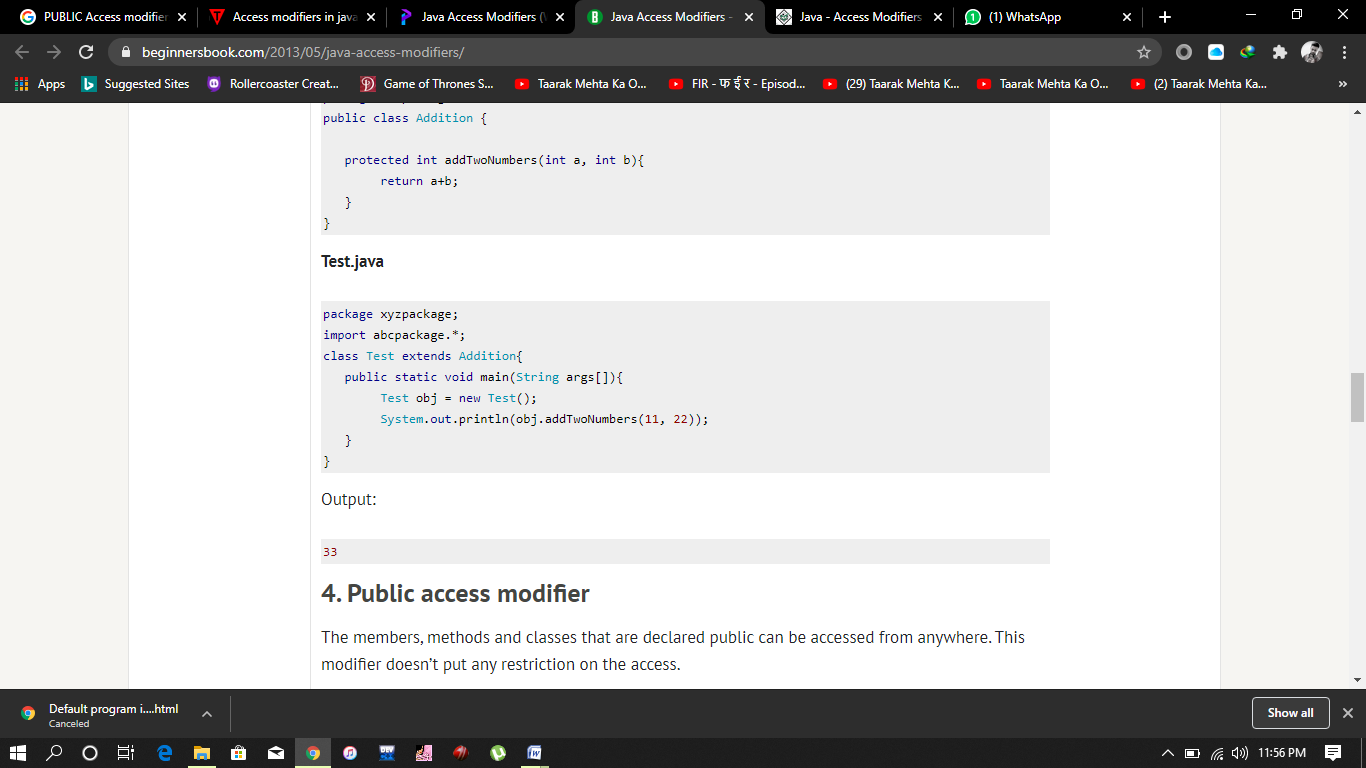
**b. Write a specific program of the above mentioned access modifiers in java.**

**ANSWER:**

* **Program in Public Access Modifier:**

**OUTPUT:**

* **Program in Protected Access Modifier:**

**OUTPUT:**

**Q3. a. What is inheritance and why it is used, discuss in detail?**

**ANSWER:**

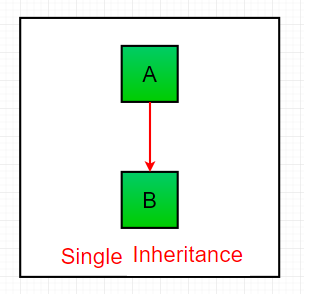
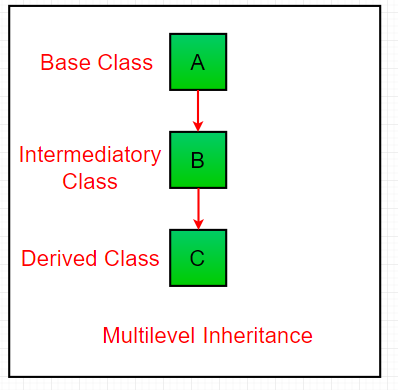
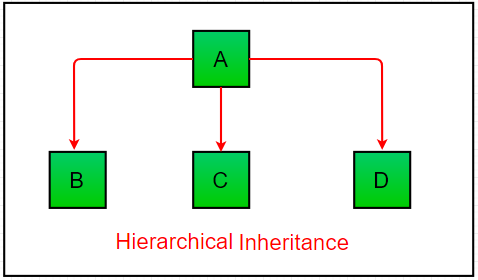
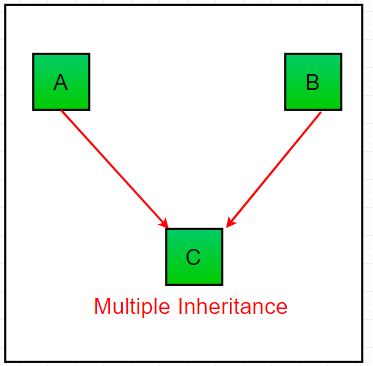
* **INHERITANCE:**

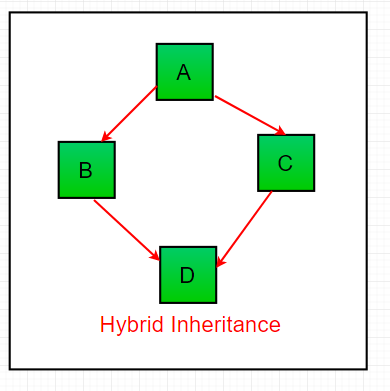
Inheritance is a mechanism in which one class acquires the property of another class. For example, a child inherits the traits of his/her parents. With inheritance, we can reuse the fields and methods of the existing class. Hence, inheritance facilitates Reusability and is an important concept of OOPs. Object-oriented systems take this a step further and allow classes to be defined in terms of other classes. For example, mountain bikes, road bikes, and tandems are all kinds of bicycles. In object-oriented terminology, mountain bikes, road bikes, and tandems are all [subclasses](javascript:var%20meth=openWin;%20meth('subclass');) of the bicycle class. Similarly, the bicycle class is the [super class](javascript:var%20meth=openWin;%20meth('superclass');) of mountain bikes, road bikes, and tandems. The Object class is at the top of class hierarchy, and each class is its descendant (directly or indirectly). A variable of type Object can hold a reference to any object, such as an instance of a class or an array. Object provides behaviors that are shared by all objects running in the Java Virtual Machine. For example, all classes inherit Object's to String method, which returns a string representation of the object. The section [Managing Inheritance](http://www.iitk.ac.in/esc101/05Aug/tutorial/java/javaOO/subclasses.html) covers the Object class in detail.

**Inheritance offers the following benefits:**

* Subclasses provide specialized behaviors from the basis of common elements provided by the super class. Through the use of inheritance, programmers can reuse the code in the super class many times.
* Programmers can implement super classes called [abstract classes](javascript:var%20meth=openWin;%20meth('abstract%20class');) that define common behaviors. The abstract super class defines and may partially implement the behavior, but much of the class is undefined and unimplemented. Other programmers fill in the details with specialized subclasses.
* **TYPES OF INHERITANCE:**

Below are the different types of inheritance which is supported by Java.

* **Single Inheritance:** In single inheritance, subclasses inherit the features of one super class. In image below, the class A serves as a base class for the derived class B.
* **Multilevel Inheritance:** In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class. In below image, the class A serves as a base class for the derived class B, which in turn serves as a base class for the derived class C. In Java, a class cannot directly access the grandparent’s members.
* **Hierarchical Inheritance:** In Hierarchical Inheritance, one class serves as a super class (base class) for more than one sub classing below image, the class A serves as a base class for the derived class B, C and D.
* **Multiple Inheritances (Through Interfaces):** In Multiple inheritances, one class can have more than one super class and inherit features from all parent classes. Please note that Java does not support multiple inheritances with classes. In java, we can achieve multiple inheritances only through Interfaces. In image below, Class C is derived from interface A and B.

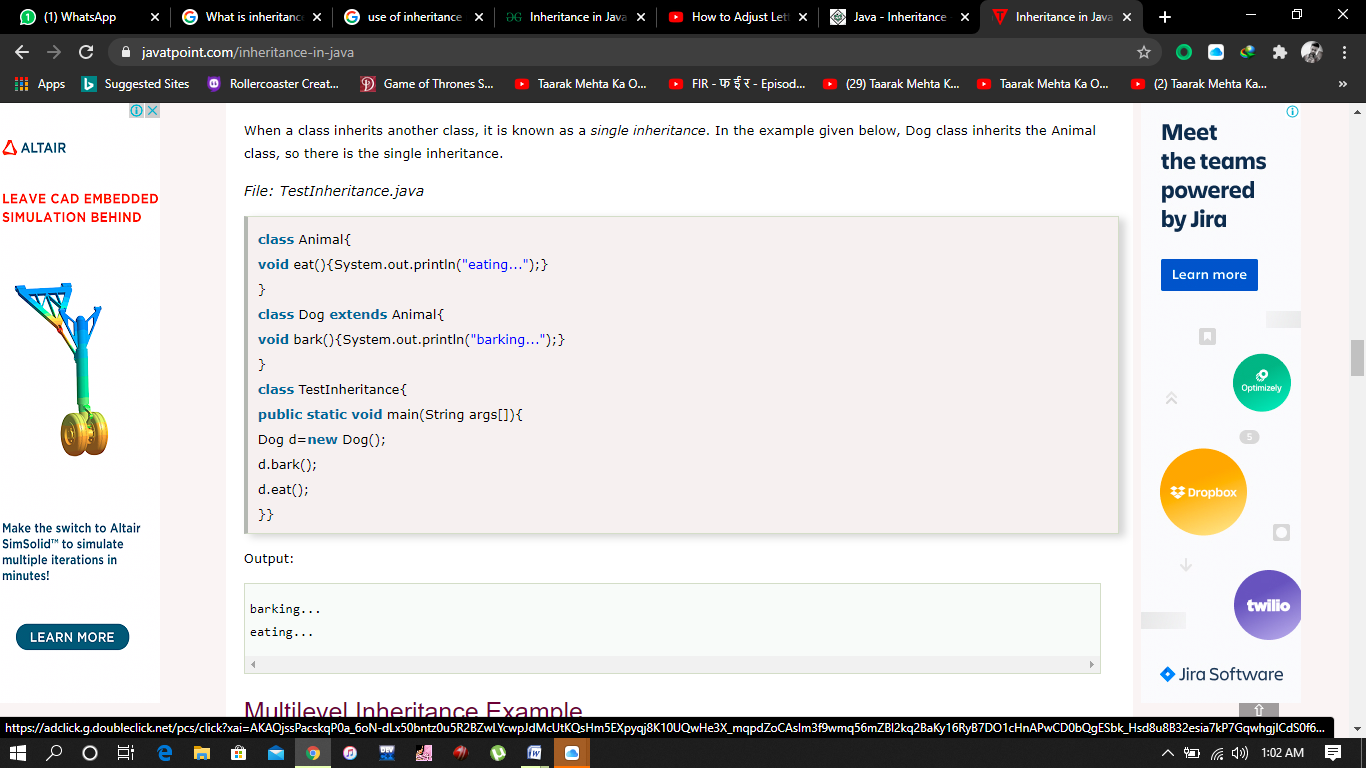
* **Hybrid Inheritance (Through Interfaces):** It is a mix of two or more of the above types of inheritance. Since java doesn’t support multiple inheritances with classes, the hybrid inheritance is also not possible with classes. In java, we can achieve hybrid inheritance only through Interfaces.

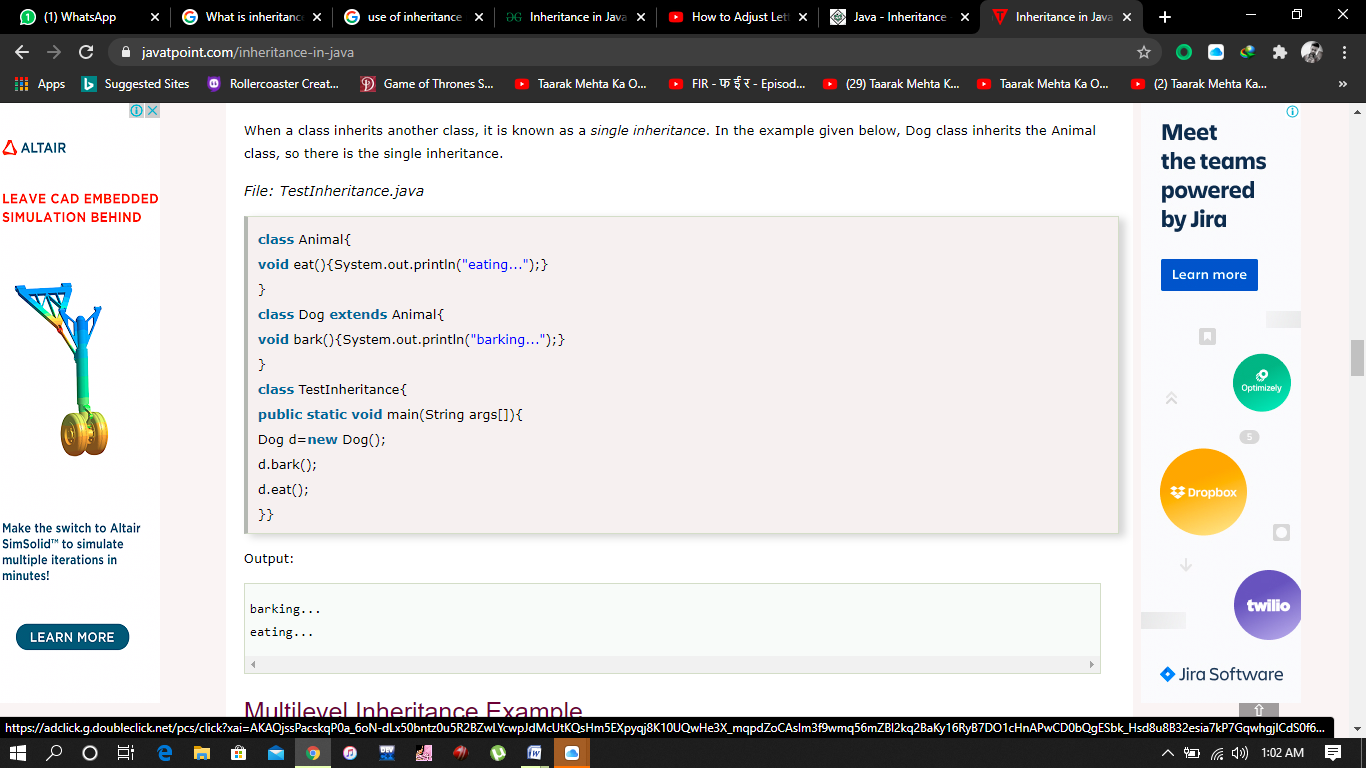
**Important facts about inheritance in Java**

* **Default super class**: Except [Object](https://www.geeksforgeeks.org/object-class-in-java/) class, which has no super class, every class has one and only one direct super class (single inheritance). In the absence of any other explicit super class, every class is implicitly a subclass of [Object](https://www.geeksforgeeks.org/object-class-in-java/) class.
* **Super class can only be one:** A super class can have any number of subclasses. But a subclass can have only **one** super class. This is because Java does not support [multiple inheritances](https://www.geeksforgeeks.org/java-and-multiple-inheritance/) with classes. Although with interfaces, multiple inheritances are supported by java.
* **Inheriting Constructors:**A subclass inherits all the members (fields, methods, and nested classes) from its super class. Constructors are not members, so they are not inherited by subclasses, but the constructor of the super class can be invoked from the subclass.
* **Private member inheritance:** A subclass does not inherit the private members of its parent class. However, if the super class has public or protected methods (like getters and setters) for accessing its private fields, these can also be used by the subclass.

**b. Write a program using Inheritance class on Animal in java.**

**ANSWER:**

* **Program using inheritance class on Animal**



**OUTPUT:**

**Q4. a. What is polymorphism and why it is used, discuss in detail?**

**ANSWER:**

* **POLYMORPHISM:**

Polymorphism is an object-oriented programming concept that refers to the ability of a variable, function or object to take on multiple forms. In a programming language exhibiting polymorphism, class objects belonging to the same hierarchical tree (inherited from a common parent class) may have functions with the same name, but with different behaviors.

**Example**

The classic example is of the Shape class and all the classes that are inherited from it, such as:

* **Rectangle**
* **Triangle**
* **Circle**

class Shape

{

  public:

  Shape(){}

  //defining a virtual function called Draw for shape class

  virtual void Draw(){cout<<"Drawing a Shape"<<endl;}

};

class Rectangle: public Shape

{

  public:

  Rectangle(){}

  //Draw function defined for Rectangle class

  virtual void Draw(){cout<<"Drawing a Rectangle"<<endl;}

};

class Triangle: public Shape

{

  public:

  Triangle(){}

  //Draw function defined for Triangle class

  virtual void Draw(){cout<<"Drawing a Triangle"<<endl;}

};

class Circle: public Shape

{

  public:

  Circle(){}

  //Draw function defined for Circle class

  virtual void Draw(){cout<<"Drawing a Circle"<<endl;}

};

int main() {

  Shape \*s;

  Triangle tri;

  Rectangle rec;

  Circle circ;

  // store the address of Rectangle

  s = &rec;

  // call Rectangle Draw function

  s->Draw();

  // store the address of Triangle

  s = &tri;

  // call Traingle Draw function

  s->Draw();

  // store the address of Circle

  s = &circ;

  // call Circle Draw function

  s->Draw();

  return 0;

}

**Explanation:**

In the example above,

We used virtual keyword while defining the Draw () functions as a virtual function is a member function which when declared in the base class can be re-defined (Overridden) by the derived classes.

At run time the compiler looks at the contents of the pointer \*s.

Since, the addresses of objects of tri, rec and circ are stored in \*s the respective Draw () function is called.

As you can see, each of the child classes has a separate implementation for the function Draw (). This is how polymorphism is generally used.

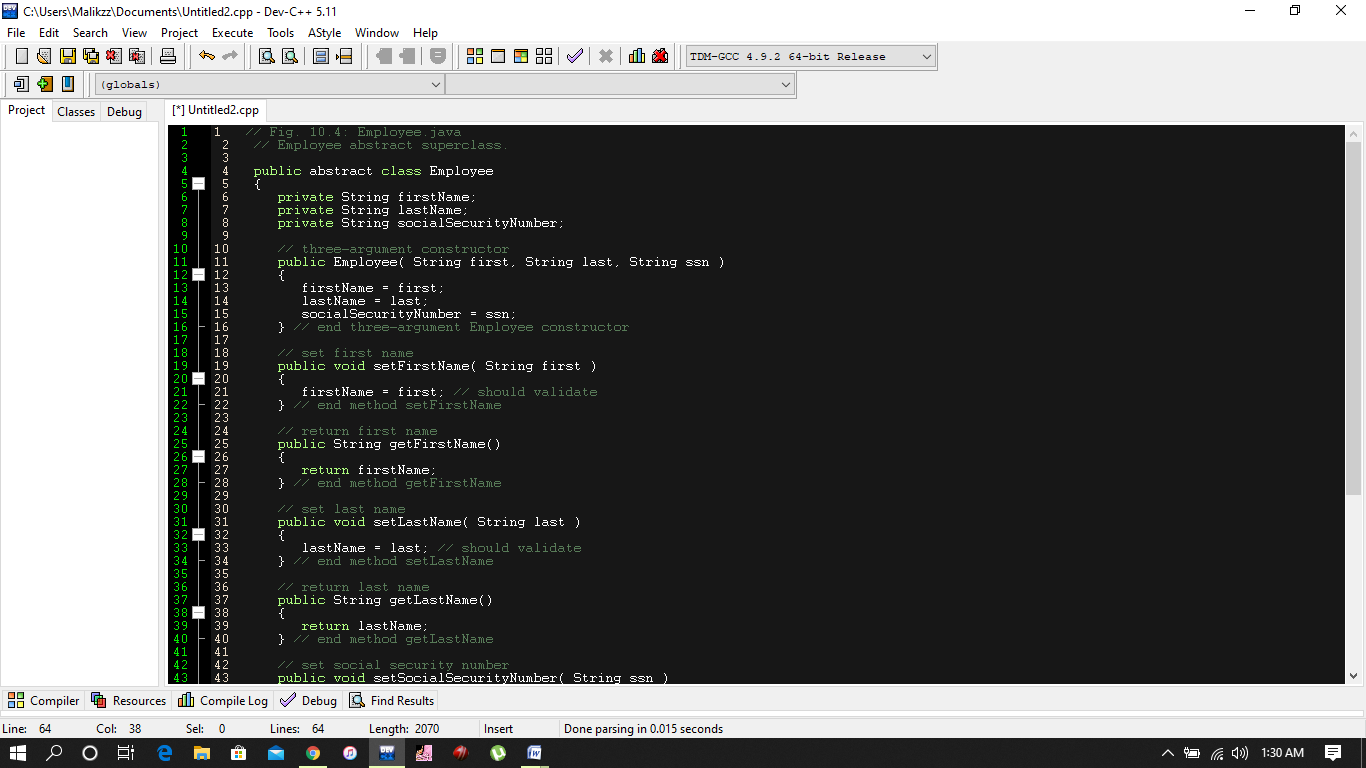
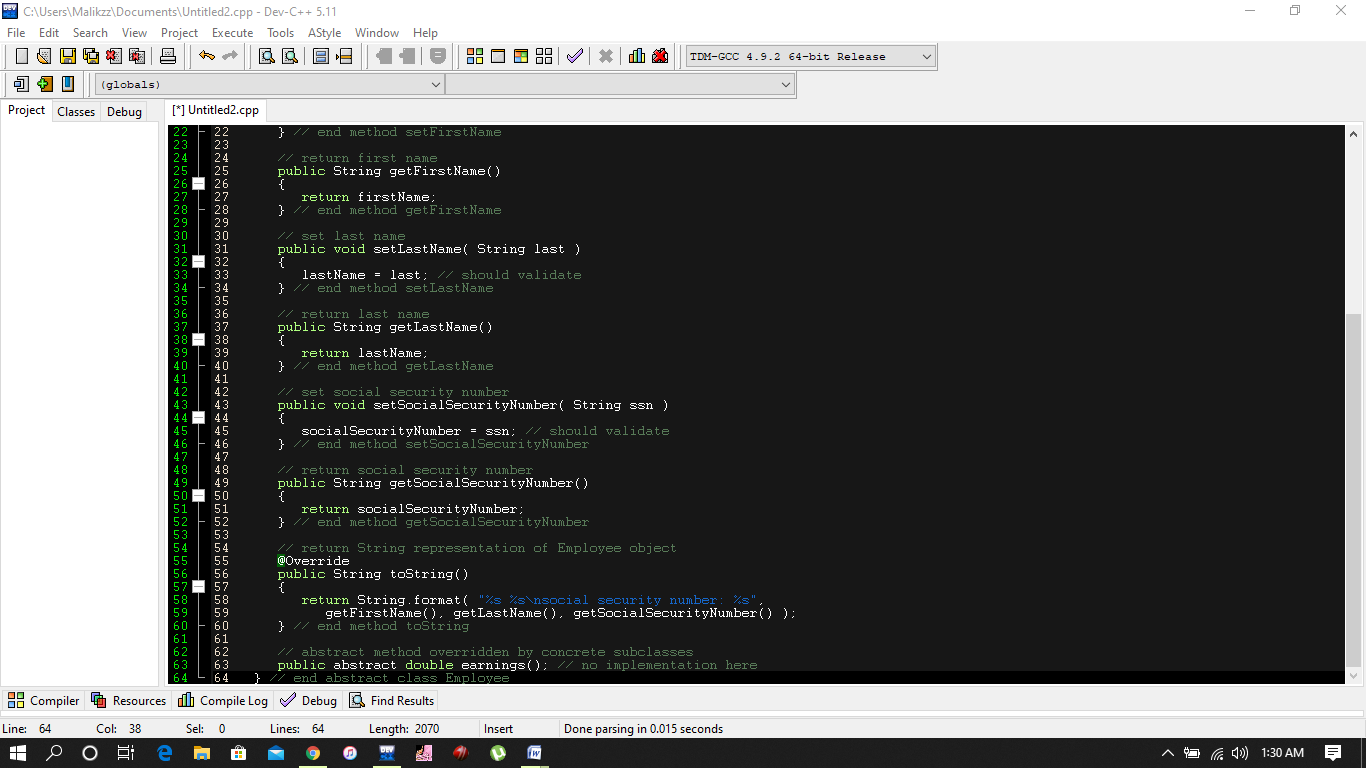
* **Types of Polymorphism**
* Compile time polymorphism
* **Example:** Method overloading
* Runtime polymorphism
* **Example:** Method overriding
* **Advantages of Polymorphism**

It helps programmers reuse code and classes once written, tested and implemented.

A single variable name can be used to store variables of multiple data types (float, double, long, int, etc).

It helps compose powerful, complex abstractions from simpler ones.

**b. Write a program using polymorphism in a class on Employee in java.**

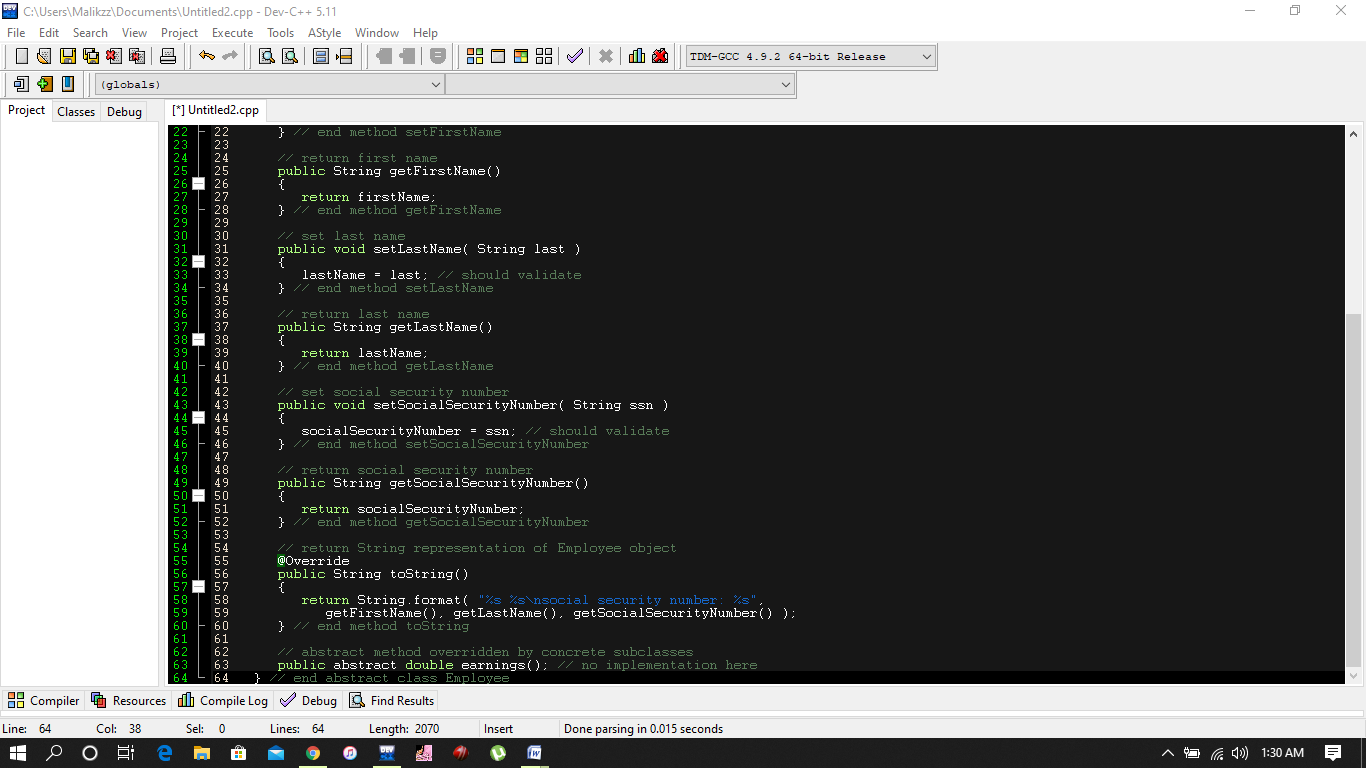
**ANSWER:**

**EXPLANATION:**

The class includes a constructor that takes the first name, last name and social security number as arguments (lines 11–16); get methods that return the first name, last name and social security number (lines 25–28, 37–40 and 49–52, respectively); set methods that set the first name, last name and social security number (lines 19–22, 31–34 and 43–46, respectively); method to String (lines 55–60), which returns the String representation of an Employee; and abstract method earnings (line 63), which will be implemented by each of the concrete subclasses. The Employee constructor does not validate its parameters in this example; normally, such validation should be provided.

**Q5. a. Why abstraction is used in OOP, discuss in detail?**

**ANSWER:**

* **ABSTRACTION IS USED IN OOP:**

Abstraction is selecting data from a larger pool to show only the relevant details of the object to the user. Abstraction “shows” only the essential attributes and “hides” unnecessary information. It helps to reduce programming complexity and effort. It is one of the most important concepts of OOPs.

**Let’s Study Abstraction concept with an Example**

Suppose you want to create a banking application and you are asked to collect all the information about your customer. There are chances that you will come up with following information about the customer but, not all of the above information is required to create a banking application. So, you need to select only the useful information for your banking application from that pool. Data like n since we have fetched/removed/selected the customer information from a larger pool, the process is referred as Abstraction. However, the same information once extracted can be used for a wide range of applications. For instance, you can use the same data for hospital application, job portal application, a Government database, etc. with little or no modification. Hence, it becomes your Master Data. This is an advantage of Abstraction.ame, address; tax information, etc. make sense for a banking application

* **What is Abstraction in Java?**

Abstraction in JAVA “shows” only the essential attributes and “hides” unnecessary details of the object from the user. In Java, abstraction is accomplished using Abstract classes, Abstract methods, and interfaces. Abstraction helps in reducing programming complexity and effort.

* **Abstract Class**

A class which is declared “abstract” is called as an abstract class. It can have abstract methods as well as concrete methods. A normal class cannot have abstract methods.

* **Abstract Method**

A method without a body is known as an Abstract Method. It must be declared in an abstract class. The abstract method will never be final because the abstract class must implement all the abstract methods.

* **Rules of Abstract Method**
* Abstract methods do not have an implementation; it only has method signature
* If a class is using an abstract method they must be declared abstract. The opposite cannot be true. This means that an abstract class does not necessarily have an abstract method.
* If a regular class extends an abstract class, then that class must implement all the abstract methods of the abstract parent

**Difference between Abstraction and Encapsulation**

|  |  |
| --- | --- |
| **Abstraction** | **Encapsulation** |
| Abstraction solves the issues at the design level. | Encapsulation solves it implementation level. |
| Abstraction is about hiding unwanted details while showing most essential information. | Encapsulation means binding the code and data into a single unit. |
| Abstraction allows focusing on what the information object must contain | Encapsulation means hiding the internal details or mechanics of how an object does something for security reasons. |

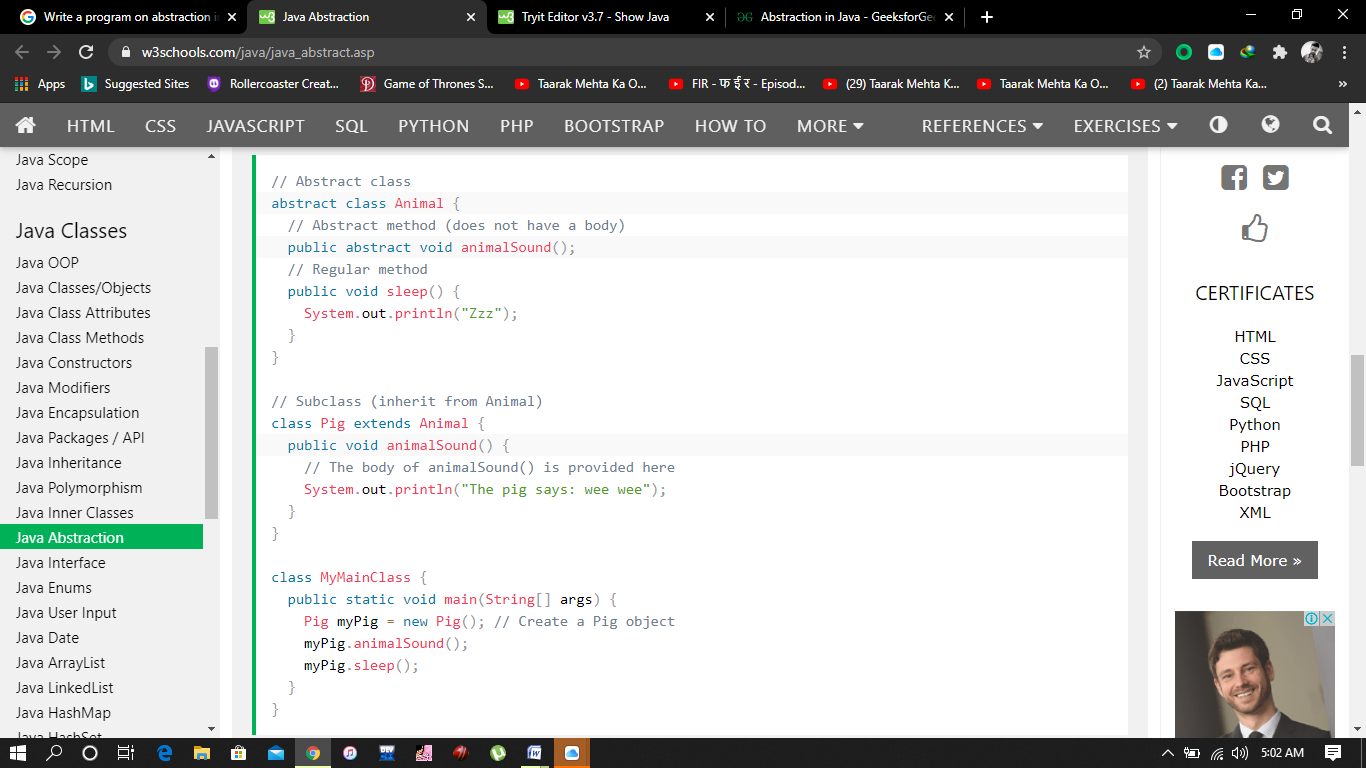
* **Advantages of Abstraction**
* The main benefit of using an abstract class is that it allows you to group several related classes as siblings.
* Abstraction helps to reduce the complexity of the design and implementation process of software.
* **When to use Abstract Methods & Abstract Class?**

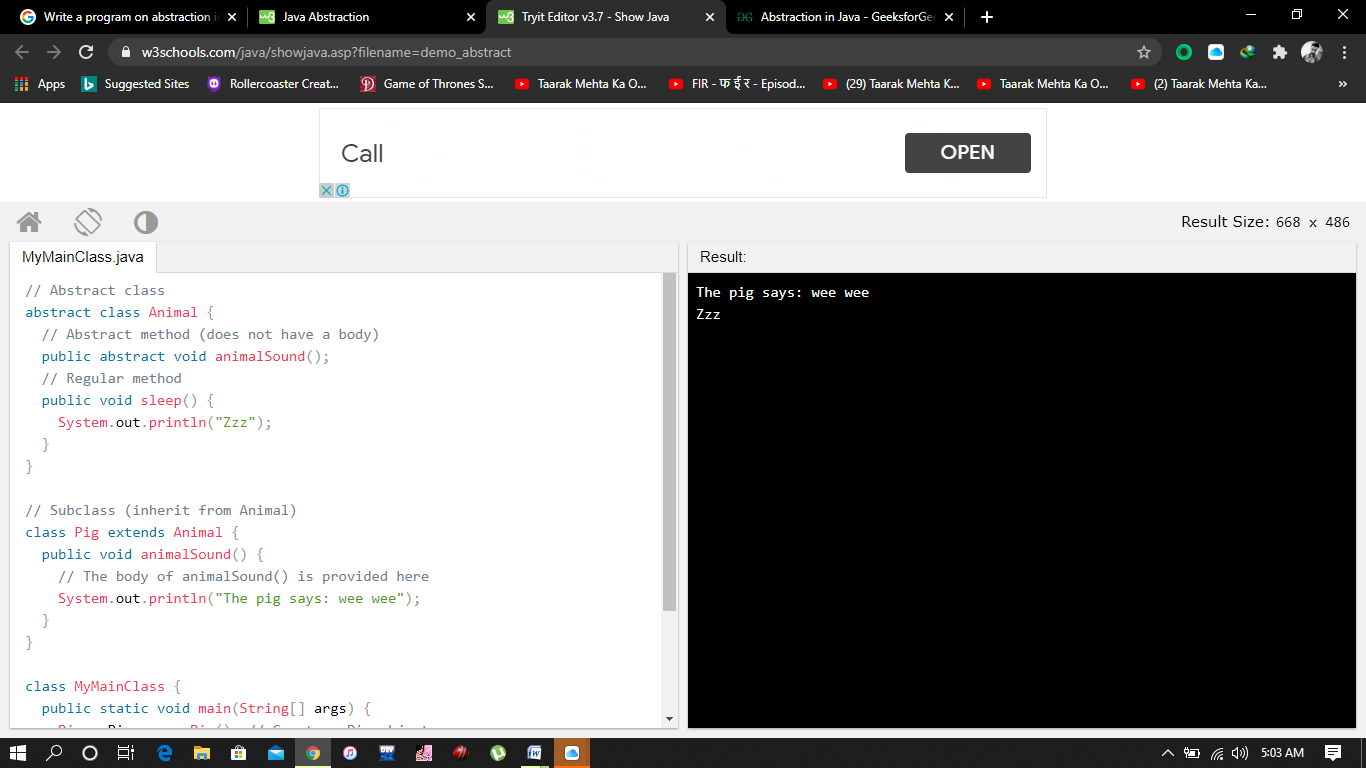
Abstract methods are mostly declared where two or more subclasses are also doing the same thing in different ways through different implementations. It also extends the same Abstract class and offers different implementations of the abstract methods. Abstract classes help to describe generic types of behaviors and object-oriented programming class hierarchy. It also describes subclasses to offer implementation details of the abstract class.

* **Summary:**
* Abstraction is the process of selecting important data sets for an Object in your software and leaving out the insignificant ones.
* Once you have modeled your object using Abstraction, the same set of data could be used in different applications.
* Java, abstraction is accomplished using Abstract classes and interfaces. We will study in detail about Abstract classes and interfaces in further tutorials.

**b. Write a program on abstraction in java.**

**ANSWER:**

* **PROGRAME ON ABSTRACTION IN JAVA:**

**OUTPUT:**

**EXPLANATION:**

Data abstraction is the process of hiding certain details and showing only essential information to the user.

Abstraction can be achieved with either abstract classes or interfaces (which you will learn more about in the next chapter).

The abstract keyword is a non-access modifier, used for classes and methods:

* **Abstract class:** is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
* **Abstract method:** can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).