



IQRA National University, Peshawar

Name	Muhammad hashim khan
Id	16001
Subject	oops
Teacher	M ayub khan
Samester	2nd
Assignment	final Assignment

Q no 1: why access modifier are used in java , explain in detail private and default access modifiers?

Ans : access modifier :

You must have seen public, private and protected keywords while practising java programs, these are called access modifiers. An access modifier restricts the access of a class, constructor, data member and method in another class. In java we have four access modifiers:

1. default
2. private
3. protected
4. Public

1. Default access modifier

When we do not mention any access modifier, it is called default access modifier. The scope of this modifier is limited to the package only. This means that if we have a class with the default access modifier in a package, only those classes that are in this package can access this class. No other class outside this package can access this class. Similarly, if we have a default method or data member in a class, it would not be visible in the class of another package. Lets see an example to understand this:

Default Access Modifier Example in Java

To understand this example, you must have the knowledge of [packages in java](#).

In this example we have two classes, Test class is trying to access the default method of Addition class, since class Test belongs to a different package, this program would throw compilation error, because the scope of default modifier is limited to the same package in which it is declared.

Addition.java

```
package abcpackage;

public class Addition {
    /* Since we didn't mention any access modifier here, it would
       * be considered as default.
       */
    int addTwoNumbers(int a, int b){
        return a+b;
    }
}
```

Test.java

```

package xyzpackage;

/* We are importing the abcpackage
 * but still we will get error because the
 * class we are trying to use has default access
 * modifier.
 */
import abcpackage.*;
public class Test {
public static void main(String args[]){
    Addition obj = new Addition();
/* It will throw error because we are trying to access
 * the default method in another package
 */
    obj.addTwoNumbers(10, 21);
}
}

```

Output:

```

Exception in thread "main" java.lang.Error: Unresolved compilation problem:
The method addTwoNumbers(int, int) from the type Addition is not visible
at xyzpackage.Test.main(Test.java:12)

```

Private access modifier

The scope of private modifier is limited to the class only.

1. Private Data members and methods are only accessible within the class
2. Class and Interface cannot be declared as private
3. If a class has private constructor then you cannot create the object of that class from outside of the class.

Let's see an example to understand this:

Private access modifier example in java

This example throws compilation error because we are trying to access the private data member and method of class ABC in the class Example. The private data member and method are only accessible within the class.

```

class ABC{
private double num =100;
private int square(int a){
    return a*a;
}
}
public class Example{
public static void main(String args[]){

```

```
ABC obj =new ABC();
System.out.println(obj.num);
System.out.println(obj.square(10));
}
}
```

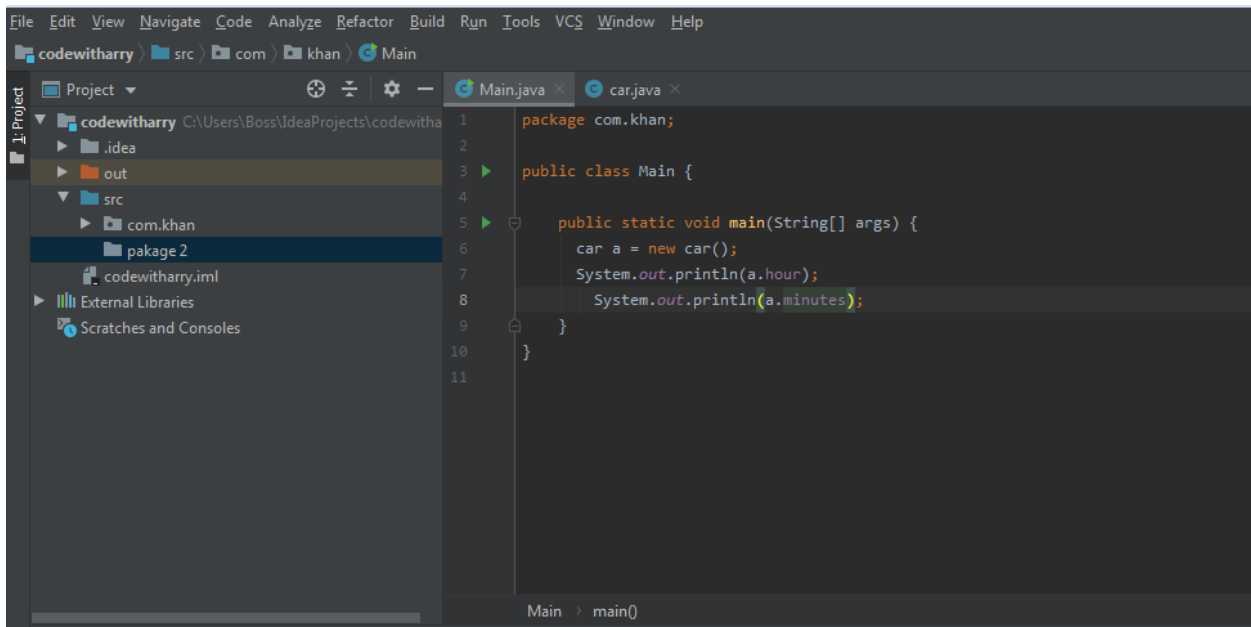
Output:

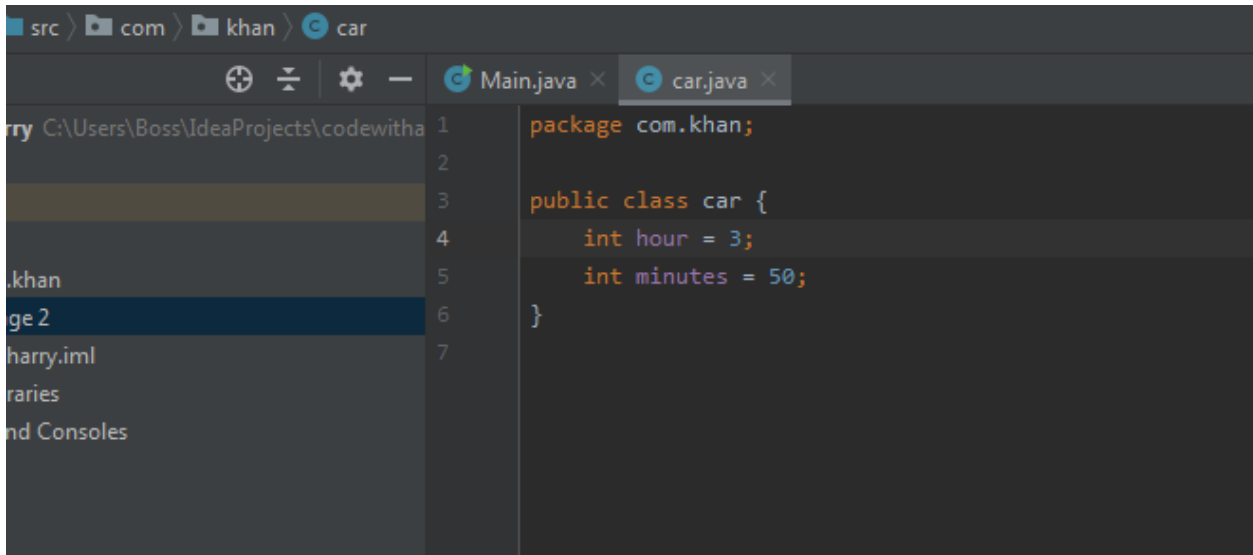
Compile- time error

B.

Write a specific program of the above mentioed access modifier in java?

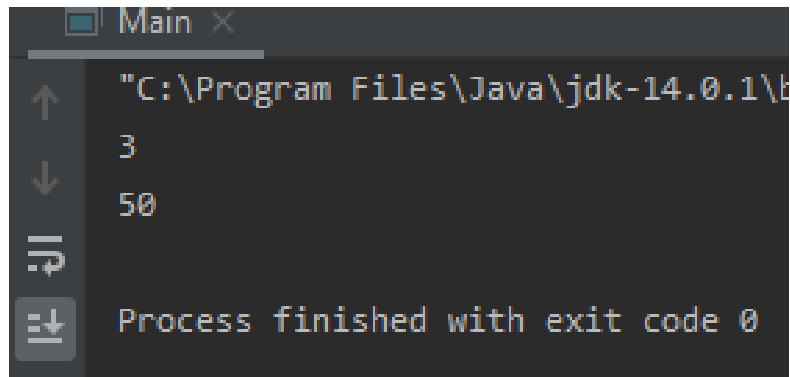
Ans: THIS IS THE DEFAULT ACCESS PROGRAM





```
src > com > khan > car
Main.java x car.java x
1 package com.khan;
2
3 public class car {
4     int hour = 3;
5     int minutes = 50;
6 }
7
```

This is the output



```
Main x
"C:\Program Files\Java\jdk-14.0.1\bin\java.exe" -Djava.class.path=C:\Program Files\Java\jdk-14.0.1\lib\java.class.path com.khan.car
3
50
Process finished with exit code 0
```

Q no 2 : explain in detail public and protected access modifier?

Ans:

Protected Access Modifier

Protected data member and method are only accessible by the classes of the same package and the subclasses present in any package. You can also say that the protected access modifier is similar to default access modifier with one exception that it has visibility in sub classes.

Classes cannot be declared protected. This access modifier is generally used in a parent child relationship.

Protected access modifier example in Java

In this example the class Test which is present in another package is able to call the addTwoNumbers() method, which is declared protected. This is because the Test class extends class Addition and the protected modifier allows the access of protected members in subclasses (in any packages).

Addition.java

```
package abcpackage;

public class Addition {

    protected int addTwoNumbers(int a, int b){

        return a+b;

    }

}
```

Test.java

```
package xyzpackage;

import abcpackage.*;

class Test extends Addition{

    public static void main(String args[]){

        Test obj = new Test();

        System.out.println(obj.addTwoNumbers(11, 22));

    }

}
```

Output:

33

Public access modifier

The members, methods and classes that are declared public can be accessed from anywhere. This modifier doesn't put any restriction on the access.

public access modifier example in java

Lets take the same example that we have seen above but this time the method addTwoNumbers() has public modifier and class Test is able to access this method without even extending the Addition class. This is because public modifier has visibility everywhere.

Addition.java

```
package abcpackage;
```

```
public class Addition {
```

```
    public int addTwoNumbers(int a, int b){
```

```
        return a+b;
```

```
    }
```

```
}
```

Test.java

```
package xyzpackage;
```

```
import abcpackage.*;
```

```
class Test{
```

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

```
        Addition obj = new Addition();
```

```
        System.out.println(obj.addTwoNumbers(100, 1));
```

```
    }
```

```
}
```

Output:

101

B. Write a specific program of the above mentioned access modifier in java?

Ans: this is the protected access modifier program

```

1  package geek;
2  import package1.*;
3  import package2.*;
4  public class Geekyshows
5  {
6      public static void main (String args[])
7      {
8          Mobile objm = new Mobile ();
9          objm.dispm ();
10         HP objh = new HP ();
11         objh.disph ();
12     }
13 }

```

```

1  package package1;
2  public class Mobile
3  {
4      public int a = 10;
5      public void dispm ()
6      {
7          Laptop objl = new Laptop ();
8          System.out.println("Mobile Class from package1 A = "+(a+objl.b));
9      }

```

```

1  package package1;
2  public class Laptop
3  {
4      protected int b = 20;
5      public void displ ()
6      {
7          System.out.println("Laptop Class from package1 B = "+b);
8      }
9  }
10 }

```



```

Geekyshows.java | Mobile.java | Laptop.java | Dell.java | Computer.java | HP.java
1  package package2;
2  import package1.*;
3  public class Computer
4  {
5      public int d = 20;
6      public void dispC ()
7      {
8          Laptop obj12 = new Laptop();
9          System.out.println("Computer Class from package2 D = "+(d+obj12.b));
10     }
11 }
12
13

```

```

Geekyshows.java | Mobile.java | Laptop.java | Dell.java | Computer.java | HP.java
1  package package2;
2  import package1.*;
3  public class HP extends Laptop
4  {
5      public int e = 40;
6      public void dispH ()
7      {
8          System.out.println("HP Class from package2 E = "+(e+b));
9      }
10 }
11

```

This is the output of this program

```

Output - Youtube (run)
run:
Mobile Class from package1 A = 30
HP Class from package2 E = 60
BUILD SUCCESSFUL (total time: 0 seconds)

```

Q no 5: why abstraction is used in opps discuss in detail ?

Ans:

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 focussing on what the information object must contain	Encapsulation means hiding the internal details or mechanics of how an object does something for security reasons.

Difference between Abstract Class and Interface

Abstract Class	Interface
An abstract class can have both abstract and non-abstract methods.	The interface can have only abstract methods.
It does not support multiple inheritances.	It supports multiple inheritances.
It can provide the implementation of the interface.	It can not provide the implementation of the abstract class.
An abstract class can have protected and abstract public methods.	An interface can have only have public abstract methods.
An abstract class can have final, static, or static final variable with any access specifier.	The interface can only have a public static final variable.

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.

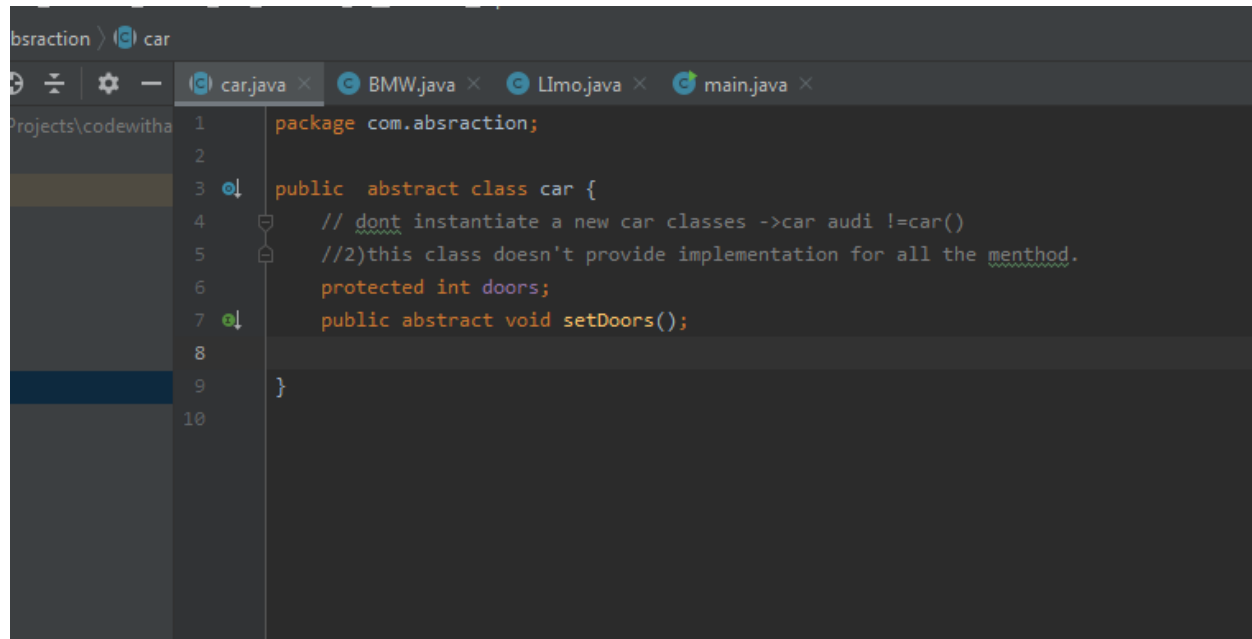
When to use Abstract Methods & Abstract Class?

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.

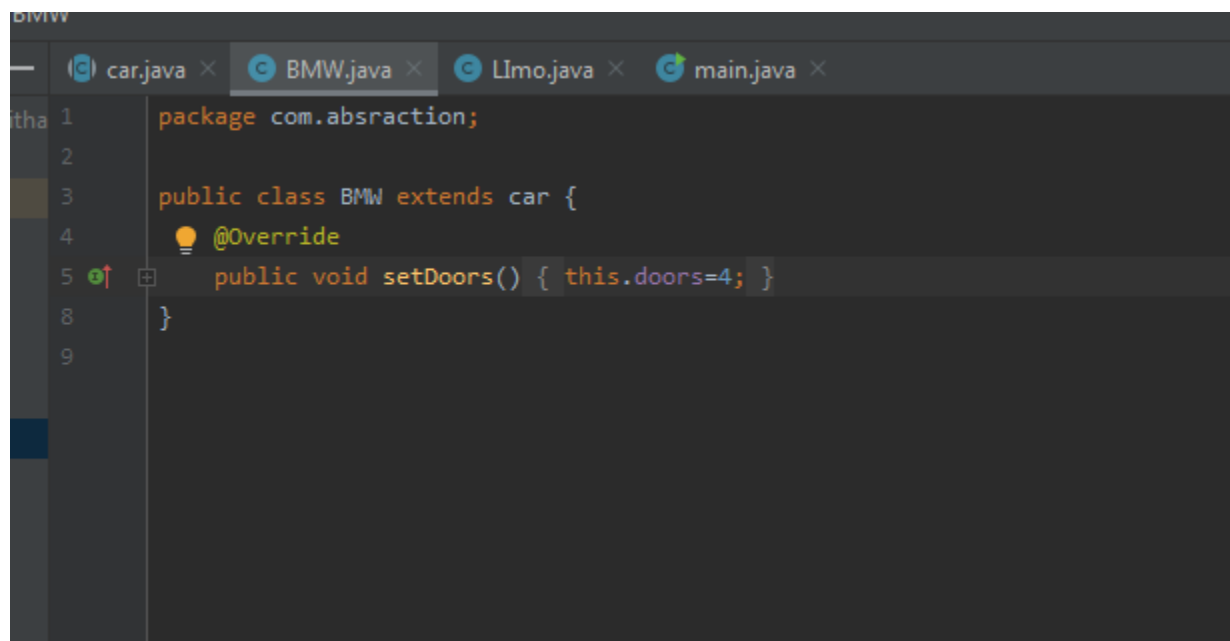
Abstract classes help to describe generic types of behaviors and object-oriented programming class hierarchy. It also describes subclasses to offer implementation details of the abstract class.

B.write a program on abstraction in java?

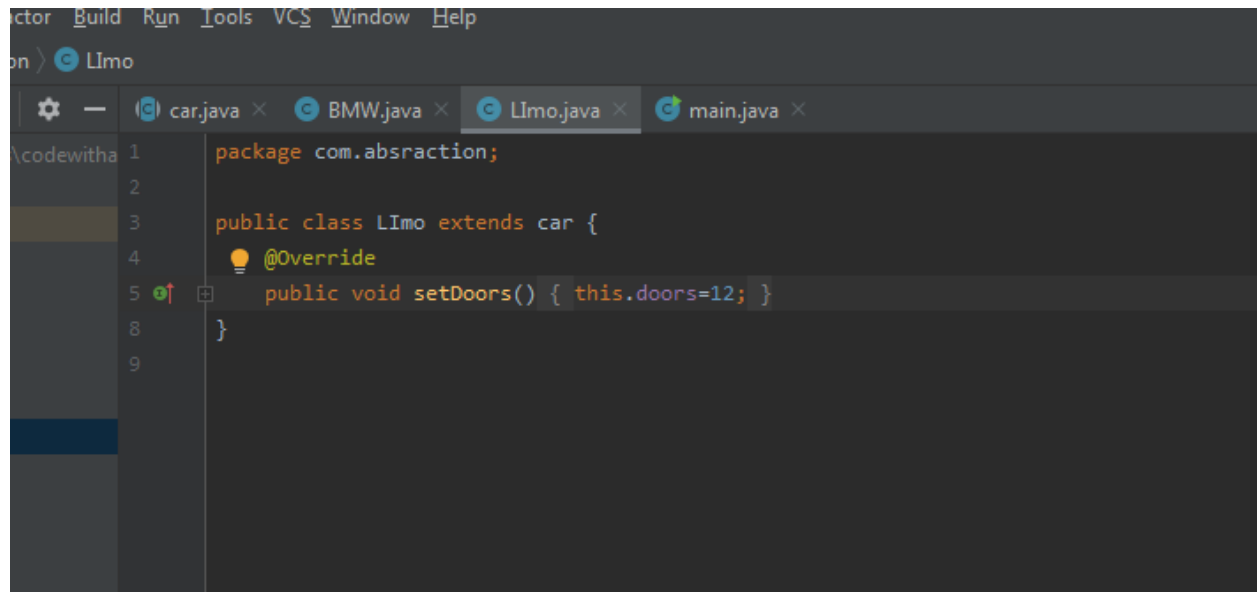
Ans:



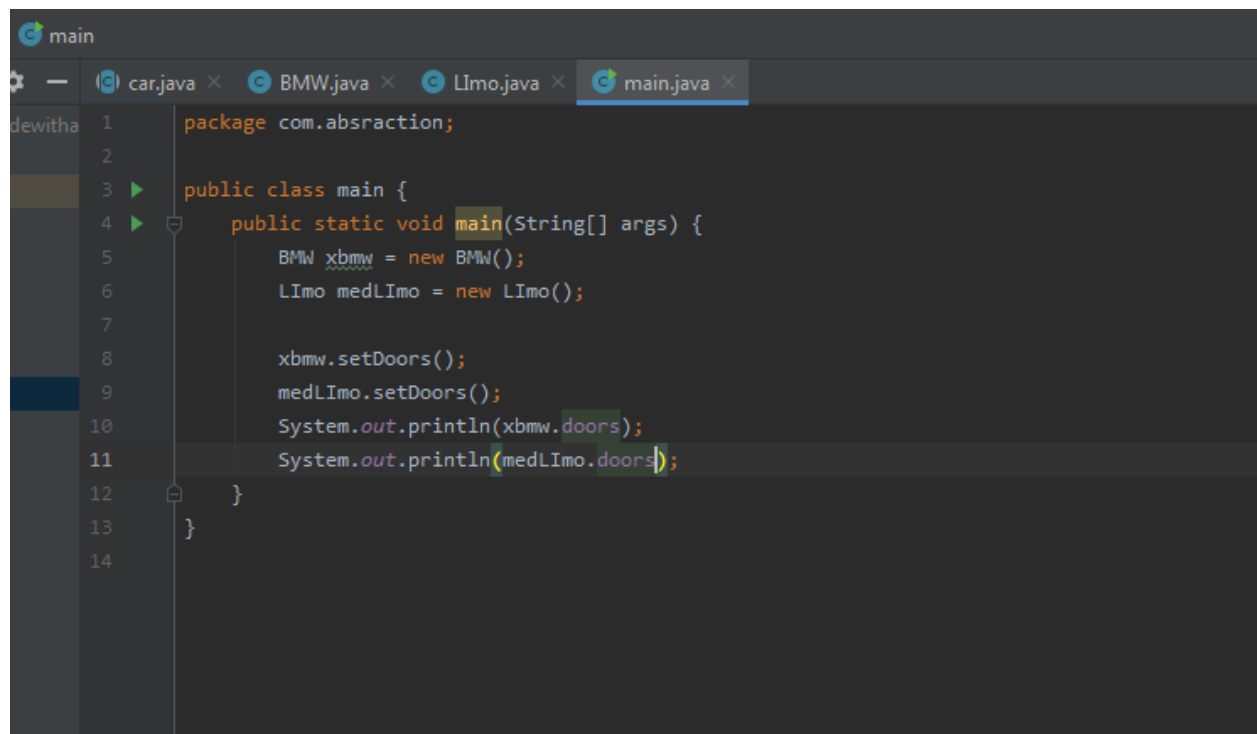
```
bsraction \ car
car.java x BMW.java x LImo.java x main.java x
1 package com.abstraction;
2
3 public abstract class car {
4     // dont instantiate a new car classes ->car audi !=car()
5     //2)this class doesn't provide implementation for all the method.
6     protected int doors;
7     public abstract void setDoors();
8
9 }
10
```



```
BMW
car.java x BMW.java x LImo.java x main.java x
1 package com.abstraction;
2
3 public class BMW extends car {
4     @Override
5     public void setDoors() { this.doors=4; }
6
7 }
8
9
```



```
ctor Build Run Tools VCS Window Help
on > LImo
car.java × BMW.java × LImo.java × main.java ×
\codewitha
1 package com.absraction;
2
3 public class LImo extends car {
4     @Override
5     public void setDoors() { this.doors=12; }
6
7
8 }
9
```



```
main
car.java × BMW.java × LImo.java × main.java ×
\codewitha
1 package com.absraction;
2
3 public class main {
4     public static void main(String[] args) {
5         BMW xbmw = new BMW();
6         LImo medLImo = new LImo();
7
8         xbmw.setDoors();
9         medLImo.setDoors();
10        System.out.println(xbmw.doors);
11        System.out.println(medLImo.doors);
12    }
13 }
14
```

THIS IS THE OUTPUT OF THIS PROGRAM

```

main x
"C:\Program Files\Java\jdk-14.0.1\bin\java.exe" "-java
4
12
Process finished with exit code 0

```

Q NO 3: What is inheritance and why it is used ,discuss in detail?

Ans :

Inheritance in Java Programming with examples

The process by which one class acquires the properties(data members) and functionalities(methods) of another class is called **inheritance**. The aim of inheritance is to provide the reusability of code so that a class has to write only the unique features and rest of the common properties and functionalities can be extended from the another class.

Child Class:

The class that extends the features of another class is known as child class, sub class or derived class.

Parent Class:

The class whose properties and functionalities are used(inherited) by another class is known as parent class, super class or Base class.

Inheritance is a process of defining a new class based on an existing class by extending its common data members and methods.

Inheritance allows us to reuse of code, it improves reusability in your java application.

Note: The biggest **advantage of Inheritance** is that the code that is already present in base class need not be rewritten in the child class.

This means that the data members(instance variables) and methods of the parent class can be used in the child class as.

If you are finding it difficult to understand what is class and object then refer the guide that I have shared on object oriented programming: [OOps Concepts](#)

Lets back to the topic:

Syntax: Inheritance in Java

To inherit a class we use extends keyword. Here class XYZ is child class and class ABC is parent class. The class XYZ is inheriting the properties and methods of ABC class.

```
class XYZ extends ABC
{
}
```

Inheritance Example

In this example, we have a base class `Teacher` and a sub class `PhysicsTeacher`. Since class `PhysicsTeacher` extends the designation and college properties and `work()` method from base class, we need not to declare these properties and method in sub class.

Here we have `collegeName`, `designation` and `work()` method which are common to all the teachers so we have declared them in the base class, this way the child classes like `MathTeacher`, `MusicTeacher` and `PhysicsTeacher` do not need to write this code and can be used directly from base class.

```
classTeacher{
String designation ="Teacher";
String collegeName ="Beginnersbook";
void does(){
    System.out.println("Teaching");
}
}

publicclassPhysicsTeacherextendsTeacher{
String mainSubject ="Physics";
publicstaticvoid main(String args[]){
    PhysicsTeacher obj =newPhysicsTeacher();
    System.out.println(obj.collegeName);
    System.out.println(obj.designation);
    System.out.println(obj.mainSubject);
    obj.does();
}
}
```

```
}
```

Output:

```
Beginnersbook  
Teacher  
Physics  
Teaching
```

Based on the above example we can say that `PhysicsTeacher` **IS-A** `Teacher`. This means that a child class has IS-A relationship with the parent class. This inheritance is known as **IS-A relationship** between child and parent class

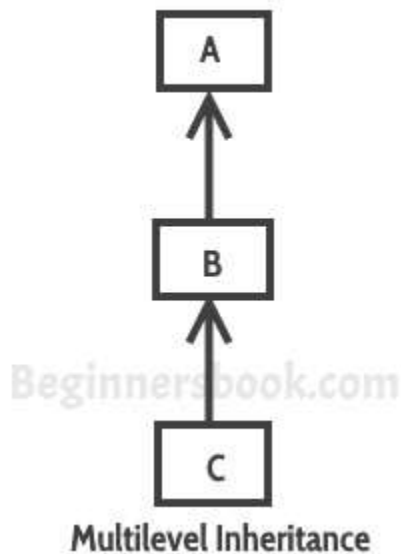
Types of inheritance

To learn types of inheritance in detail, refer: [Types of Inheritance in Java](#).

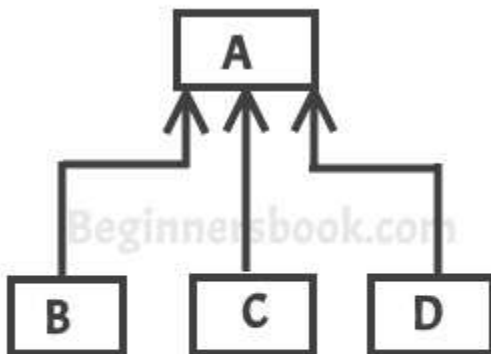
Single Inheritance: refers to a child and parent class relationship where a class extends the another class.

Multilevel inheritance: refers to a child and parent class relationship where a class extends the child class. For example class C extends class B and class B

extends class A.



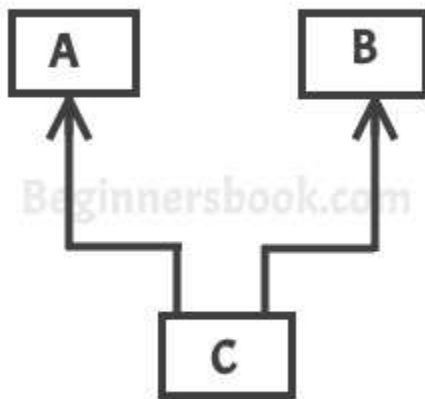
Hierarchical inheritance: refers to a child and parent class relationship where more than one classes extends the same class. For example, classes B, C & D extends the same class A.



Hierarchical Inheritance

Multiple Inheritance: refers to the concept of one class extending more than one classes, which means a child class has two parent classes. For example class C extends both classes A and B. Java doesn't support multiple inheritance,

read more about it [here](#).



Multiple Inheritance

Hybrid inheritance: Combination of more than one types of inheritance in a single program. For example class A & B extends class C and another class D extends class A then this is a hybrid inheritance example because it is a combination of single and hierarchical inheritance.

B.

Write a program using inheritance class of animal in java.

```
public class Dog extends Animal{
    private boolean ha = true;
    public void isHa(){
        if(ha){
            System.out.println("i am Ha");
        }else{
            System.out.println("i am not Ha");
        }
    }
}
```

```

public class Cat extends Animal{
    private boolean shy = true;
    public void isShy(){
        if(shy){
            System.out.println("i am shy");
        }else{
            System.out.println("i am not shy");
        }
    }
}

```

```

public class Animal {
    private String name = "joe";
    public static void main(String[] args){
        Animal a = new Animal();
        Dog d = new Dog();
        Cat c = new Cat();

        a.sayName();
        d.sayName();
        d.isHa();
        c.sayName();
        c.isShy();
    }
    public void sayName(){
        System.out.println(name);
    }
}

```

This is the output of this program

```

joe
joe
i am Ha
joe
i am shy

```

Q no .4:what is polymorphism and why is used discuss in detail?

Ans :

Polymorphism

This post provides the theoretical explanation of polymorphism with real-life examples. For detailed explanation on this topic with java programs refer [polymorphism in java](#) and [runtime & compile time polymorphism](#).

- Polymorphism means to process objects differently based on their data type.
- In other words it means, one method with multiple implementation, for a certain class of action. And which implementation to be used is decided at runtime depending upon the situation (i.e., data type of the object)
- This can be implemented by designing a generic interface, which provides generic methods for a certain class of action and there can be multiple classes, which provides the implementation of these generic methods.

Lets us look at same example of a car. A car have a gear transmission system. It has four front gears and one backward gear. When the engine is accelerated then depending upon which gear is engaged different amount power and movement is delivered to the car. The action is same applying gear but based on the type of gear the action behaves differently or you can say that it shows many forms (polymorphism means many forms)

Polymorphism could be static and dynamic both. [Method Overloading](#) is static polymorphism while, [Method overriding](#) is dynamic polymorphism.

- **Overloading** in simple words means more than one method having the same method name that behaves differently based on the arguments passed while calling the method. This called static because, which method to be invoked is decided at the time of compilation
- **Overriding** means a derived class is implementing a method of its super class. The call to overridden method is resolved at runtime, thus called runtime polymorphism

B.write a program using polymorphism in a class on employee in java?

```
/* File name : Employee.java */
public class Employee {
    private String name;
    private String address;
    private int number;

    public Employee(String name, String address, int number) {
        System.out.println("Constructing an Employee");
        this.name = name;
        this.address = address;
        this.number = number;
    }

    public void mailCheck() {
        System.out.println("Mailing a check to " + this.name + " " + this.address);
    }

    public String toString() {
        return name + " " + address + " " + number;
    }

    public String getName() {
        return name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String newAddress) {
        address = newAddress;
    }

    public int getNumber() {
        return number;
    }
}
```

Now suppose we extend Employee class as follows –

```

/* File name : Salary.java */
public class Salary extends Employee {
    private double salary; // Annual salary

    public Salary(String name, String address, int number, double salary) {
        super(name, address, number);
        setSalary(salary);
    }

    public void mailCheck() {
        System.out.println("within mailCheck of Salary class ");
        System.out.println("Mailing check to " + getName()
            + " with salary " + salary);
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double newSalary) {
        if(newSalary >= 0.0) {
            salary = newSalary;
        }
    }

    public double computePay() {
        System.out.println("Computing salary pay for " + getName());
        return salary/52;
    }
}

```

Now you study the following program carefully and try to determine its output

Now, you study the following program carefully and try to determine its output

```

/* File name : VirtualDemo.java */
public class VirtualDemo {

    public static void main(String [] args) {
        Salary s = new Salary("Mohd Mohtashim", "Ambehta, UP", 3, 3600.00);
        Employee e = new Salary("John Adams", "Boston, MA", 2, 2400.00);
        System.out.println("Call mailCheck using Salary reference --");
        s.mailCheck();
        System.out.println("\n Call mailCheck using Employee reference--");
        e.mailCheck();
    }
}

```

This is the output of this program

```
Constructing an Employee
Constructing an Employee

Call mailCheck using Salary reference --
Within mailCheck of Salary class
Mailing check to Mohd Mohtashim with salary 3600.0

Call mailCheck using Employee reference--
Within mailCheck of Salary class
Mailing check to John Adams with salary 2400.0
```