



NAME: NAVEED ALI

COURSE NAME: OBJECTED ORIENTED PROGRAMMIG

REG ID: 15958

DEPARTMENT OF BS SOFTWARE ENGINEERING

SEMESTER 2ND

Question 1:

A: Why access modifier are used in java, explain in detail. Private and Default.

Answer: There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

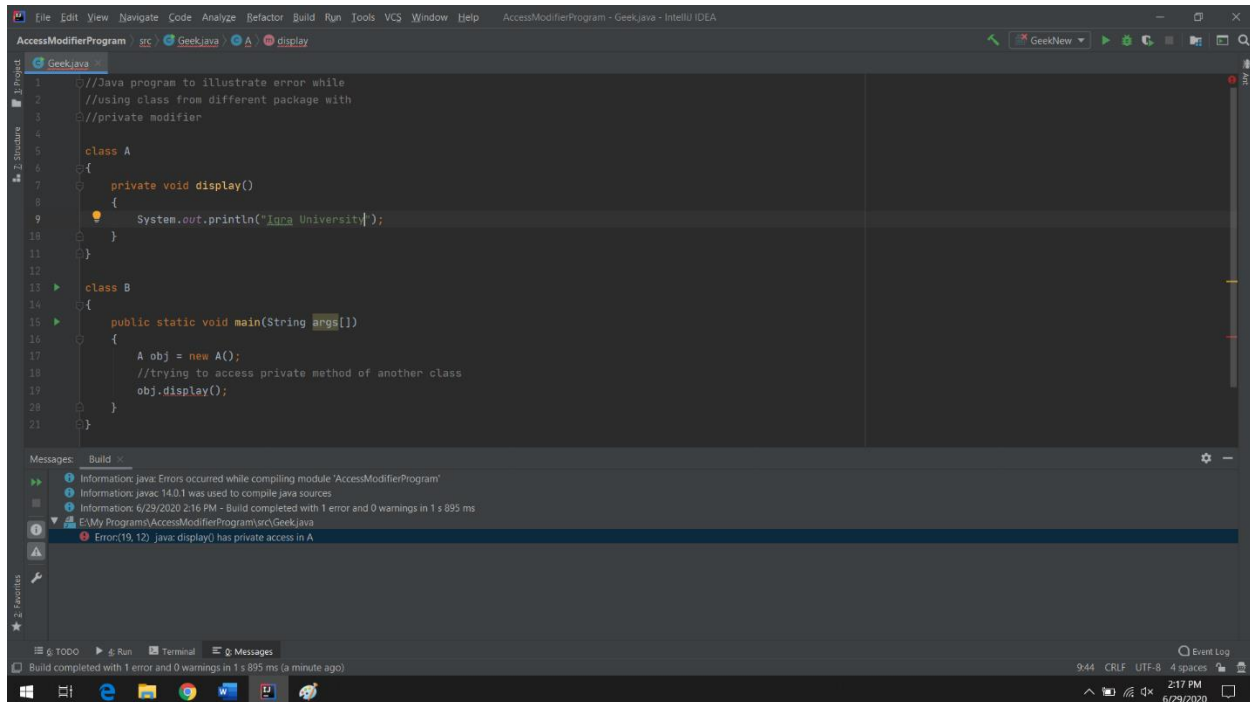
The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

1. **Private:** The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default:** The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

B: Write a specific program of the above mentioned access modifiers in java.

Private:

The private access modifier is accessible only within the class.



```
1 //Java program to illustrate error while
2 //using class from different package with
3 //private modifier
4
5 class A
6 {
7     private void display()
8     {
9         System.out.println("Iana Universit{");
10    }
11 }
12
13 class B
14 {
15     public static void main(String args[])
16     {
17         A obj = new A();
18         //trying to access private method of another class
19         obj.display();
20     }
21 }
```

Messages: Build

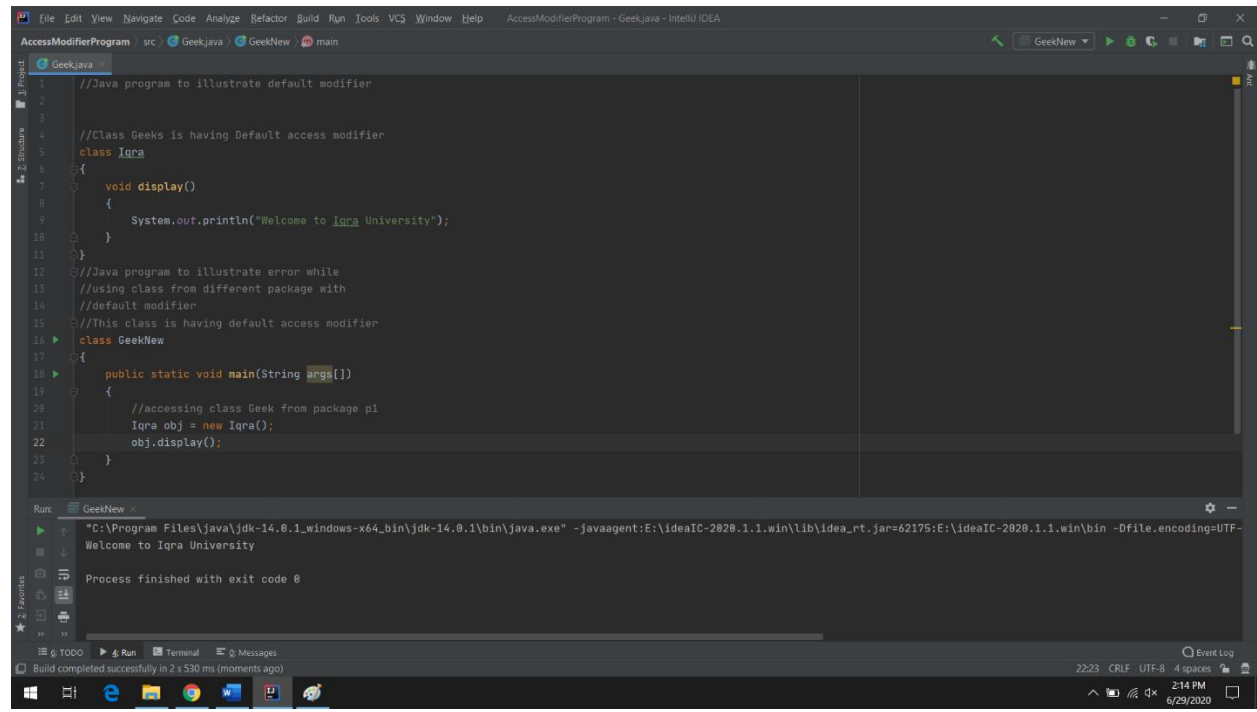
- Information: java: Errors occurred while compiling module 'AccessModifierProgram'
- Information: java: 14.0.1 was used to compile java sources
- Information: 6/29/2020 2:16 PM - Build completed with 1 error and 0 warnings in 1 s 895 ms
- Error(19, 12) java: display() has private access in A

Build completed with 1 error and 0 warnings in 1 s 895 ms (a minute ago)

9:44 CRLF UTF-8 4 spaces 2:17 PM 6/29/2020

Default:

If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package. It cannot be accessed from outside the package. It provides more accessibility than private. But it is more restrictive than protected, and public.



```
1 //Java program to illustrate default modifier
2
3
4 //Class Geeks is having Default access modifier
5 class Iqra
6 {
7     void display()
8     {
9         System.out.println("Welcome to Iqra University");
10    }
11 }
12
13 //Java program to illustrate error while
14 //using class from different package with
15 //default modifier
16 //This class is having default access modifier
17 class GeekNew
18 {
19     public static void main(String args[])
20     {
21         //accessing class Geek from package p1
22         Iqra obj = new Iqra();
23         obj.display();
24     }
25 }
```

Run: GeekNew x
"C:\Program Files\java\jdk-14.0.1\windows-x64_bin\jdk-14.0.1\bin\java.exe" -javaagent:E:\ideaIC-2020.1.1.win\lib\idea_rt.jar=62175:E:\ideaIC-2020.1.1.win\bin -Dfile.encoding=UTF-8
Welcome to Iqra University
Process finished with exit code 0

Question 2: Explain in detail Public and Protected access modifiers?

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

Answer:

A: The protected access modifier is specified using the keyword **protected**.

- The methods or data members declared as protected are **accessible within same package or sub classes in different package**.

```
//Java program to illustrate
```

```
//protected modifier
```

```
//Class A
```

```
public class A
```

```
{
```

```
    protected void display()
```

```
    {
```

```
        System.out.println("IQRA UNIVERSITY");
```

```

    }
}
class B extends A
{
    public static void main(String args[])
    {
        B obj = new B();
        obj.display();
    }
}

```

OUTPUT:

The screenshot shows an IDE window with the following content:

```

1 //Java program to illustrate
2 //protected modifier
3 //Class B is subclass of A
4 class B extends A
5 {
6     public static void main(String args[])
7     {
8         B obj = new B();
9         obj.display();
10    }
11 }
12 }
13 |

```

The Run window at the bottom shows the execution command and output:

```

Run: B x
"C:\Program Files\Java\jdk-14.0.1\windows-x64_bin\jdk-14.0.1\bin\java.exe" -javaagent:E:\ideaIC-2020.1.1.win\lib\idea_rt.jar=62395:E:\ideaIC-2020.1.1.win\bin -Dfile.encoding=UTF-8
IqRA UNIVERSITY
Process finished with exit code 0

```

Public: The public access modifier is specified using the keyword **public**.

- The public access modifier has the **widest scope** among all other access modifiers.
- Classes, methods or data members which are declared as public are **accessible from everywhere** in the program. There is no restriction on the scope of a public data members.

PROGRAM:

```
//Java program to illustrate
```

```
//public modifier
public class A
{

    public void msg()
    {
        System.out.println("IQRA UNIVERSITYT");
    }

}
class B{
    public static void main(String args[]){
        A obj = new A();
        obj.msg();
    }
}
```

OUTPUT:

The screenshot shows the IntelliJ IDEA IDE with a project named 'A - B.java - IntelliJ IDEA'. The 'Project' view on the left shows a package structure with 'src' containing 'A' and 'B'. The 'Run' view at the bottom shows the execution of 'B.java' with the output 'IQRA UNIVERSITYT' and the message 'Process finished with exit code 0'.

Question 3: What is inheritance and why it is used.

ANSWER:

Inheritance in Java

Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.

Important terminology:

- **Super Class:** The class whose features are inherited is known as super class(or a base class or a parent class).

- **Sub Class:** The class that inherits the other class is known as sub class(or a derived class, extended class, or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
- **Reusability:** Inheritance supports the concept of “reusability”, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

Write a program using Inheritance class on Animal in java.

Animal Program Screenshots:

Main Animal Class:

```

1  public class Animal
2  {
3
4      public static void main(String[] args)
5      {
6          Dogs obj=new Dogs();//DOG OBJECTS
7          obj.setDogcolour("brown");
8          obj.setDogage(12);
9          obj.setDogbreed("BullDog");
10         obj.setAllinfo();
11         System.out.println();
12
13         Cat obj1=new Cat();
14         obj1.setCatcolor("Black");
15         obj1.setCatage(6);
16         obj1.setCatbreed("Persian cat");
17         //inherited class info
18         obj1.setDogcolour("white");
19         obj1.setDogage(18);
20         obj1.setDogbreed("German Shepard");
21         //output
22         obj1.setMoreInfo();
23         System.out.println();
24         obj1.setAllinfo();
25     }
26 }
27 }
28

```

Dog Class:

The screenshot shows the IntelliJ IDEA IDE with the 'Dogs.java' file open. The code defines a 'Dogs' class with private attributes for color, age, and breed, and public methods to set these values and print all information.

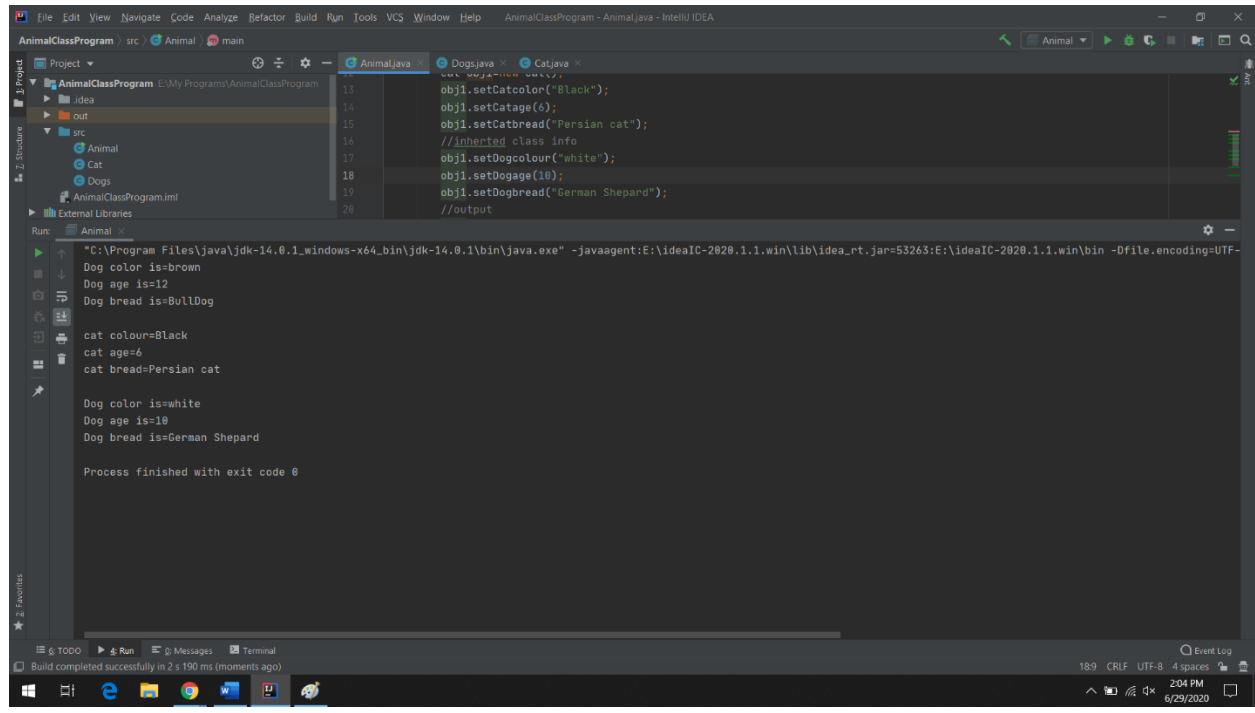
```
1 public class Dogs
2 {
3     private String dogcolour;
4     private int dogage;
5     private String dogbreed;
6
7     public Dogs()
8     {
9
10    }
11    public void setDogcolour(String dogcolour) { this.dogcolour=dogcolour; }
12    public void setDogage(int dogage) { this.dogage=dogage; }
13
14    public void setDogbreed(String dogbreed) { this.dogbreed = dogbreed; }
15    public void setAllInfo()
16    {
17        System.out.println("Dog color is="+dogcolour);
18        System.out.println("Dog age is="+dogage);
19        System.out.println("Dog breed is="+dogbreed);
20    }
21 }
22
23
24
25
26
27
28
29
30
31
```

Cat Class extends Dog Class:

The screenshot shows the IntelliJ IDEA IDE with the 'Cat.java' file open. The code defines a 'Cat' class that extends the 'Dogs' class, with its own private attributes and public methods.

```
1 public class Cat extends Dogs
2 {
3     private String catcolor;
4     private int catage;
5     private String catbreed;
6
7     public Cat() { super(); }
8
9     public void setCatcolor(String catcolor)
10    {
11        this.catcolor = catcolor;
12    }
13
14    public void setCatage(int catage) { this.catage = catage; }
15    public void setCatbreed(String catbreed) { this.catbreed=catbreed; }
16    public void setMoreInfo()
17    {
18        System.out.println("cat colour="+catcolor);
19        System.out.println("cat age="+catage);
20        System.out.println("cat breed="+catbreed);
21    }
22 }
23
24
25
26
27
28
29
30
31
32
```

OUTPUT:



Question 4: What is polymorphism and why it is used.

Answer:

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object.

Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object.

It is important to know that the only possible way to access an object is through a reference variable. A reference variable can be of only one type. Once declared, the type of a reference variable cannot be changed.

The reference variable can be reassigned to other objects provided that it is not declared final. The type of the reference variable would determine the methods that it can invoke on the object.

A reference variable can refer to any object of its declared type or any subtype of its declared type. A reference variable can be declared as a class or interface type.

B: Write a program using polymorphism in a class on Employee in java.

Program:

```
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 gate");  
    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;
}
}
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);
    }

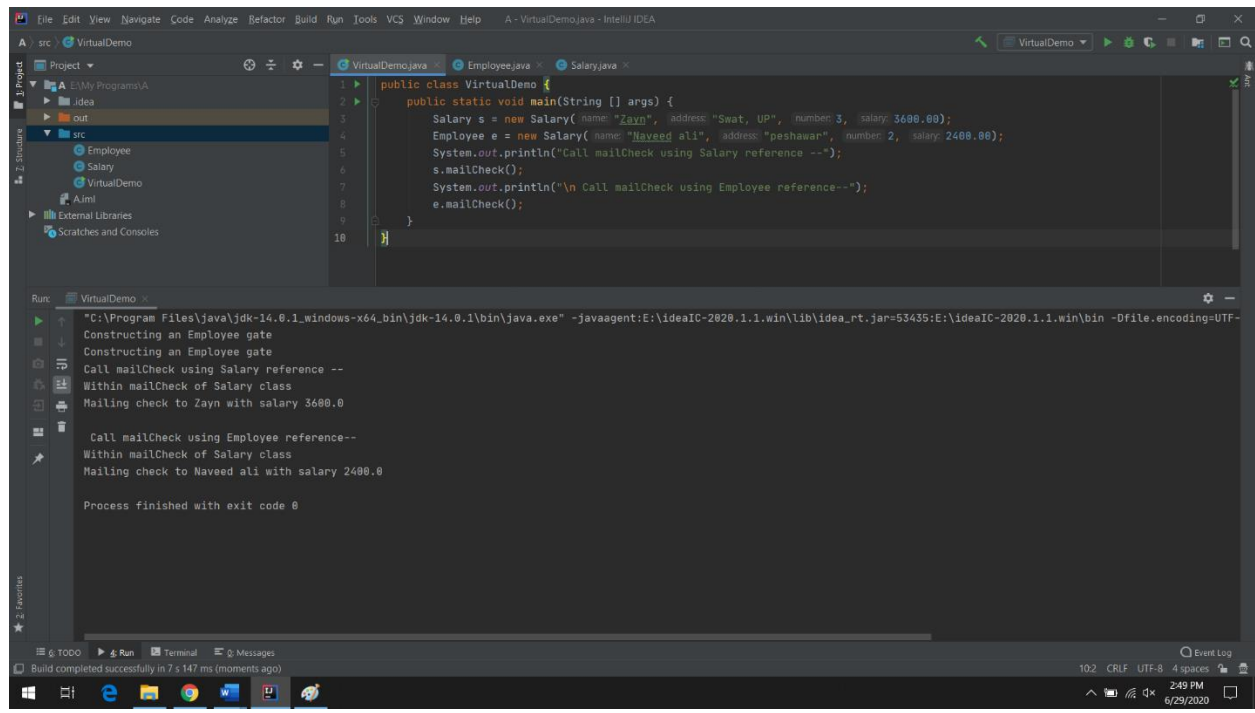
    public void mailCheck() {
        System.out.println("Within mailCheck of Salary class ");
        System.out.println("Mailing check to " + getName()
            + " with salary " + salary);
    }

    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;
}
}
public class VirtualDemo {
    public static void main(String [] args) {
        Salary s = new Salary("Zayn", "Swat", 3, 3600.00);
        Employee e = new Salary("Naveed ali", "peshawar", 2, 2400.00);
        System.out.println("Call mailCheck using Salary reference --");
        s.mailCheck();
        System.out.println("\n Call mailCheck using Employee reference--");
        e.mailCheck();
    }
}
```

OUTPUT:



Question: Why abstraction is used in OOP, discuss in detail?
Write a program on abstraction in java.

Answer:

What is Abstraction 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.

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

Program:

```
// Abstract class

abstract class Animal {

    // Abstract method (does not have a body)

    public abstract void animalSound();

    // Regular method

    public void sleep() {

        System.out.println("Zzz");

    }

}

// Subclass (inherit from Animal)

class Cat extends Animal {

    public void animalSound() {

        // The body of animalSound() is provided here

        System.out.println("The Cat says: meow meow");

    }

}

class MyMainClass {

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

```
Cat myCat = new Cat(); // Create a Pig object
```

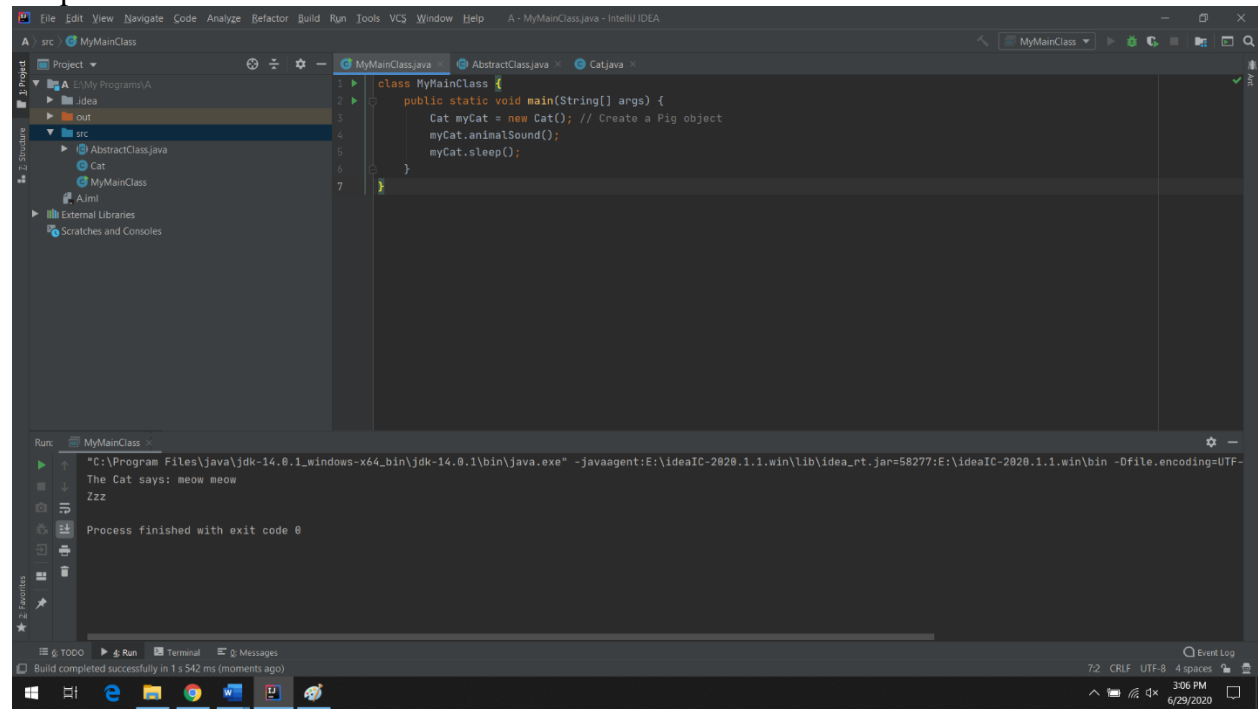
```
myCat.animalSound();
```

```
myCat.sleep();
```

```
}
```

```
}
```

Output:



```
class MyMainClass {  
    public static void main(String[] args) {  
        Cat myCat = new Cat(); // Create a Pig object  
        myCat.animalSound();  
        myCat.sleep();  
    }  
}
```

Run: MyMainClass

```
"C:\Program Files\java\jdk-14.0.1\windows-x64_bin\jdk-14.0.1\bin\java.exe" -javaagent:E:\ideaIC-2020.1.1.win\lib\idea_rt.jar=58277:E:\ideaIC-2020.1.1.win\bin -Dfile.encoding=UTF-8  
The Cat says: meow meow  
ZZZ  
Process finished with exit code 0
```

Build completed successfully in 1 s 542 ms (moments ago)

7:2 CRLF UTF-8 4 spaces 3:06 PM 6/29/2020