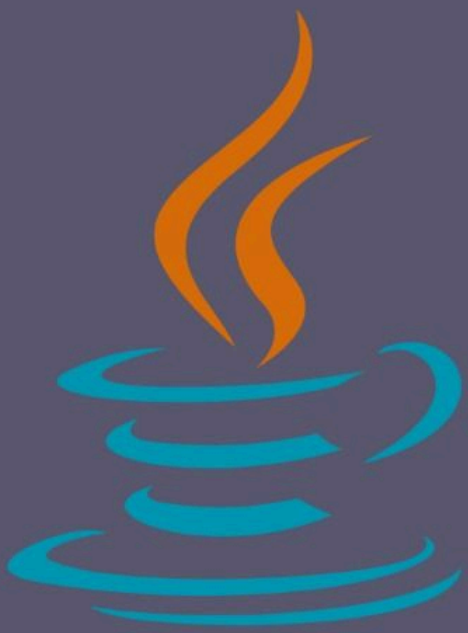
OBJECT ORIENTED PROGRAMMING USING



Java

Let's explore technology
together to live in the future

Checkout more on
https://github.com/Sy-hash-collab

Sy-hash-collab

26/9/23

Modifiers:

Used to set access level for → classes, attributes, methods, constructors

1) Access Modifier: controls access level

1.1) Class

never use private keyword.

→ **Public**: The public class is accessible by any other class.

→ **Default / Package**: This class is only accessible by classes in the same package.

* package is a collection of related classes.

// Public

Teacher
TeacherName TeacherSalary

```
public class Teacher
{
  accessible to
} both packages
```

```
public class Main
{
  public static void
     main (String[]a)
  {System.out.println
          ("Hello");
  }
```

// Default

Student
StudentAge StudentName

```
class Student
{
  accessible to this
} package only.
```

```
class MyClass
{
public static void main
     (String[]args)
{System.out.println("D"
```

1.2) Attributes, Methods, Constructors:

- **public** : The code is accessible for all classes.

```
public class Main
{
   public String name = "John";
   public int age = 24;
}
class Second
{  public static void main (String [] args)
   {  Main myObj = new Main();
      System.out.println ("Name:" + myObj.name);
      System.out.println ("Age:" + myObj.age);
   }
}
```

- **default** : The code is accessible in the same package only.

```
Class Person
{
   String name = "John";
   int age = 24;

  public static void main (String [] args)
   {  Person myObj = new Person();
System.out.println ("Name:" + myObj.name);
System.out.println ("Age:" + myObj.age);
}
```

- **private**: The code is only accessible within the declared class

```
public class Main
{ private String name= "John";
  private int age = 24;
}
public static void main (String [] args)
{ Main myObj = new Main();
System.out.println ("Name: " + myObj.name
System.out.println ('Age: "+ myObj.age);
}
```

- **protected**: The code is accessible in the same package and subclasses.

```
class Person
{ protected String name = "John";
  protected int age = 24;
}
class Student extends Person
{
   private int marks = 40;
public static void main (String [] args)
  { Student = myObj = new Student();
System.out.println ( "Name: " + myObj.name;
System.out.println ( "Age: " + myObj.age );
System.out.println ("Marks: "+ myObj.marks);
}}
```

```
Name: John
Age: 24
Marks: 40
```

If we make a child class of the Teacher class and place this child class inside another class i-e Student class then this childclass will have access to the protected members of the Teacher class.

```
class Teacher → class
{
  protected String name = "John";
  protected int age = 24;
}
class Student → another class
{  class Supervisor extends Teacher → sub-class
   {  public static void main (String [] a)
      {  Supervisor s = new Supervisor();
System.out.println ("Name: " + name);
System.out.println ("Age: " + age);
      }
   }
}
```

Note: The methods or data members declared as protected are
- accessible
- within same package.
- subclasses within different packages

2) Non-Access Modifiers: donot control access-level, provides another functionality

2.1) Class: For classes, you can use:

- final: The class cannot be inherited by other classes, can't be extended to sub-class (child-class).

```
final class Test  // final class.
{ void  mNumber()
{ System.out.println ("31304");
}
  void  atmPIN()
{ System.out.println ("8944);
}}
class Thief extends Test  // sub-class
{
  void  mNumber()  // This will produce erro
{ System.out.println ("231304");
}
  void  atmPIN()// This will produce error
{ System.out.println ("8945);
}}
class Final
{ public static void main (String [Jargs)
{   Thief t = new Theif();
    t.mNumber(); t.atmPIN();
}
```

abstract: A class which contains the abstract keyword in its decleration is abstract class.

Rules:

=> The abstract class "cannot be used to create objects."

e.g

```
abstract class Animal // abstract class, can't be instantiated
{
}
class Demo
{ public static void main (String [] args)
  {
    Animal a = new Animal();
  }
}
```

=> We can make a 'reference' of Animal class. If we make a "sub-class / child class" of superclass Animal then we can access this "abstract class Animal" and can also store the object of its child class as a reference in it.

```
abstract class Animal
{ // body of abstract class is provided by its child / sub-class.
}
class Dog extends Animal
{
}
class Demo
{ public static void main (String [] args)
  {
    Animal a = new Dog(); // making an object of class Dog() and storing it in abstract class Animal
  }
}
```

=> It may or maynot contain abstract methods

It can have abstract and non-abstract methods.

To use an abstract class, you have to inherit it from sub-classes.

If a class contain partial implementation then we should declare a class as abstract.

```
class B → we should declare this class as abstract class bez it has partial implementation if doesn't matter if its object is created or not.
{
  public void m1(),
    {
      // no implementation
    }
}
```

If a class contains abstract method then we must declare it as abstract class.

```
class A → this class must be declared as abstract.
{
  abstract void m1();

}
```

When there's a same behaviour of two or more classes then we make an abstract class for these classes, which will contain the behaviour/method that's similar between the two classes. And we make these two classes, the subclasses of abstract class so that the abstract class can access the subclasses objects.

```
abstract class Animal // abstract class can access objects of its sub-classes
{
 public abstract void eat (); // abstract method

}

class Dog extends Animal // Sub-class
{
}

class Tiger extends Animal // sub-class
{
}

class Demo
{
 public static void main (String[]args)
  {
   Animal a = new Dog (); // creating
   Animal b = new Tiger(); objects of child classes of Animal class
  }
}
```

```
abstract class Animal // abstract class can access objects of its sub-classes
{
 public abstract void eat (); // abstract method

}

class Dog extends Animal // Sub-class
{
}

class Tiger extends Animal // sub-class
{
}

class Demo
{
 public static void main (String[]args)
  {
   Animal a = new Dog (); // creating
   Animal b = new Tiger(); objects
                           of child classes
                           of Animal class
  }
}
```

2.2) Attributes / Variables

- **final:** final keyword indicates that the specific attribute or variable can't be changed or modified.

```
public class Main
{ public static void main (String [] args)
    {
        final double PI = 3.14159;
      System.out.println(PI);
        PI = 4.5; // This line will give an error.
    }
}
```

Output : 3.14159

- **Static:** static keyword means that the entity to which it is applied is available outside any particular instance of class. That means "static methods and attributes" are part of class and not an object. The memory is allocated to such an attribute or method at the time of class loading.

A static field exist across all the class instances and without creating an object of class they can be called.

```
class static_gfg
{   //static variable
    static String s = "Geeks for Geeks";
}
class GFG
{ public static void main (String [] args)
    { // no object required
      System.out.println (static_gfg.s);
  }
```

2.3) Methods

- **final** : final means methods can't be overridden / modified.

```
class final-gfg
{
  final void myMethod()
  {
    System.out.println("Geeks for Geeks");
  }
class override_final_gfg extends final-gfg
  {
    void myMethod()  // trying to override
                     method
    {
      System.out.println("Override Geeks
                          for Geek");
    }
  }
  class GFG
{
  public static void main(String []args)
{
  override_final_gfg obj = new override_final
                           -gfg();
    obj.myMethod();
}}
```

myMethod() in "final_gfg" class is declared as final, and we're trying to override that from the "over_ride_final_gfg" class. So, it's producing error here.

- **static:** A static method means that it can be accessed without creating an object of the class, unlike public.

```
public class Main
{
    // static method
  static void myStaticMethod()
  {
    System.out.println("Static method is
                called without creating
                objects");
  }
    // public method
  public void myPublicMethod()
  {
    System.out.println("Public method
                must be called by
                creating objects");
  // Main method
  public static void main(String[] args)
  {
      myStaticMethod(); // call static method

      Main myObj = new Main(); // creating
                object of Main
    myObj.myPublicMethod(); // call public
                method
  }
}
```

Output:

Static method is called without creating objects.
Public method must be called by creating objects.

• abstract : A method which contain abstract modifier at the time of decleration is called abstract method.

Rules

. It can only be used in abstract class.
. It "doesn't contain any body" and always ends with ";".

abstract <return type> <method name> ( ) ;

e.g abstract void f1 ( ) ;

=> Whenever the action is common but implementation is different then we should use abstract method.

```
abstract class fruit
{
    abstract void taste();
```

Abstract method must be overridden in sub-classes otherwise, it will also become a abstract class.

Any subclass needs to be either implement all the methods of abstract class, or it should also need to be an abstract class.

The abstract keyword cannot be used with static, final or private becauze the prevent overriding and we need to override methods in case of abstract class.

```
abstract class abstract_gfg
{
    // abstract method
    abstract void myMethod();
    // no body
}
// sub-class of abstract class
class Myclass extends abstract_gfg
{
    // overriding the abstract method
    void myMethod()
    {
        System.out.println("Geeks for Geeks");
        // body of abstract method provided
        by sub-class
    }
}
class GFG
{
    public static void main(String[] args)
    {
        Myclass myObj = new Myclass();
        obj.myMethod();
    }
}
```