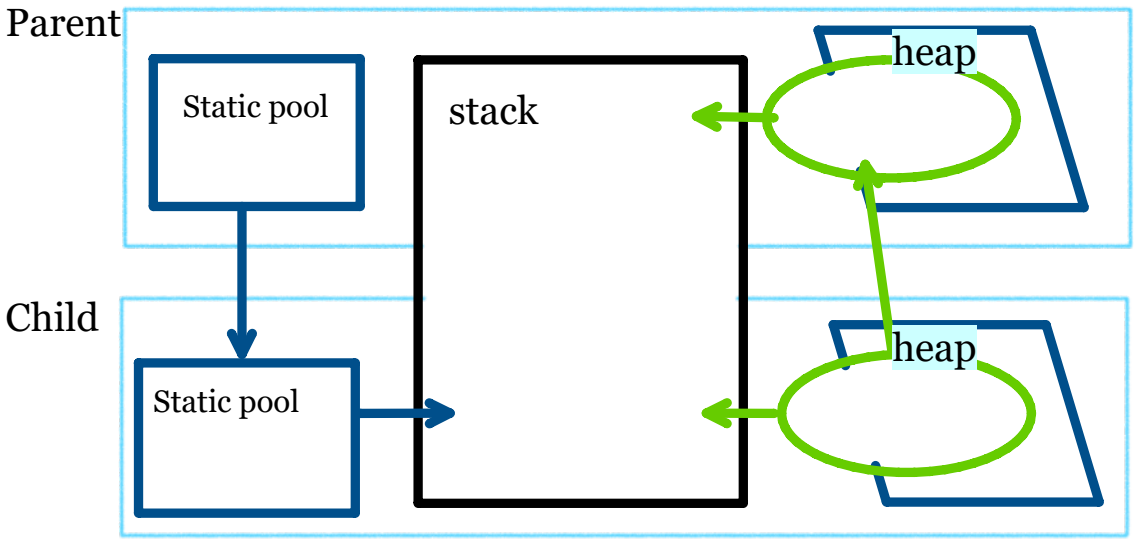
Types of inheritance

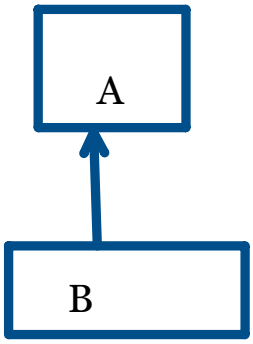
While performing inheritance, we use both the parent and child class. Which means, here both classes are going to get memory, and both the instances are created.



According the above diagram, we can understand how a child class can access the parent data.

1. <u>Single level:</u>

   Here, we have one child class extending properties and the behaviors of a single parent class.




```
class A
{
   //data//
}
class B extends A
{
   //data//
   //parent A data//
}
```

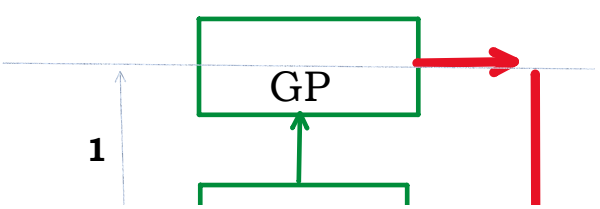
```
class Google
{
   String username, password;
```

```
  public Google(){}
  public Google(String username, String password)
  {
    this.username=username;
    this.password=password;
  }
  {
    System.out.println("Account is created");
  }
  public boolean login(String u, String p)
  {
    if( username== u && password==p)
    {
      System.out.println("Login Successfull");
      return true;
    }
    else
    {
      System.out.println("Login failed");
      return false;
    }
  }
}

class Gmail extends Google
{
  public Gmail(){}
  public Gmail(String username, String password)
  {
    super(username, password);
  }
  {
    System.out.println("Gmail account verified");
  }

  public void send_recieve_Mail()
  {
    System.out.println(username+" can send or recieve emails");
  }
}

class User//driver
{
  public static void main(String[] args)
  {
    Gmail user1=new Gmail("Qspider","qwerty");
    if(user1.login("Qspider", "qwerty"))
    {
      user1.send_recieve_Mail();
    }
  }
}
```

2. Multilevel inheritance:

Here, we can have n number of parent and child classes, which are in is a relationship in multiple levels.

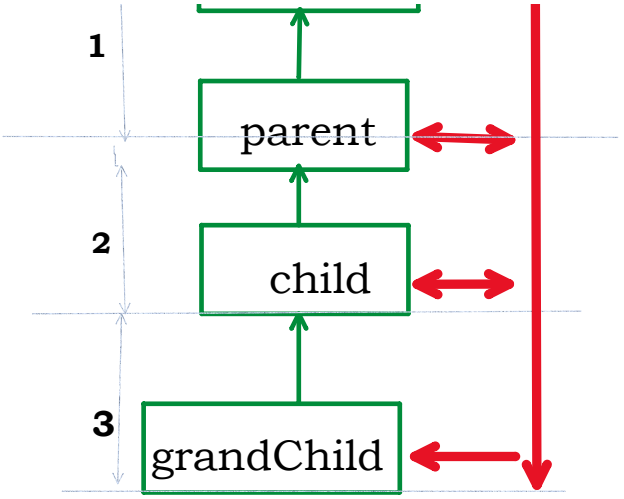
```
Class grandparent
{
}

Class parent extends grandparent
```



Class parent extends grandparent
{
}

Class child extends parent
{
}

Class grandchild extends child
{
}



The lowermost class will have the data of all the upper classes.

Note:
The lowermost class is called as base *class.*
The uppermost class is called as super *class.*

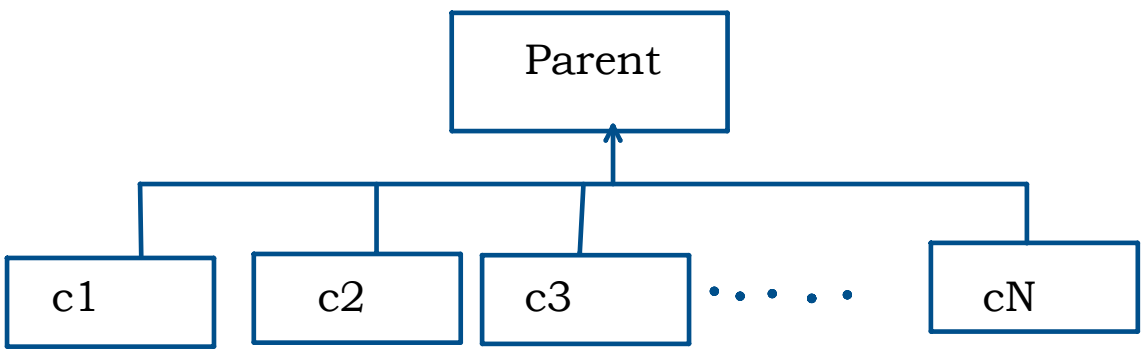
```
class Living//Supermost Class
{
    String def="Living beings";
    public Living()
    {
        System.out.println(this.def);
    }
}

class Mammals extends Living
{
    String def="Those who have spine";
    public Mammals()
    {
        super();
        System.out.println(this.def);
    }
}
class Dog extends Mammals
{
    String def="Man's best friend";

    public Dog()
    {
        super();
        System.out.println(this.def);
```

```
    }
}
//base class
class puppy extends Dog
{
    String def="cute";
    public puppy()
    {
        super();
        System.out.println(this.def);
    }
}
public class human//driver
{
    public static void main(String[] args)
    {
        puppy p=new puppy();
    }
}
```

```
[Running] cd "c:\Programming\OOP\" && javac human.java && java human

Living beings
Those who have spine
Man's best friend
Cute

[Done] exited with code=0 in 2.242 seconds
```

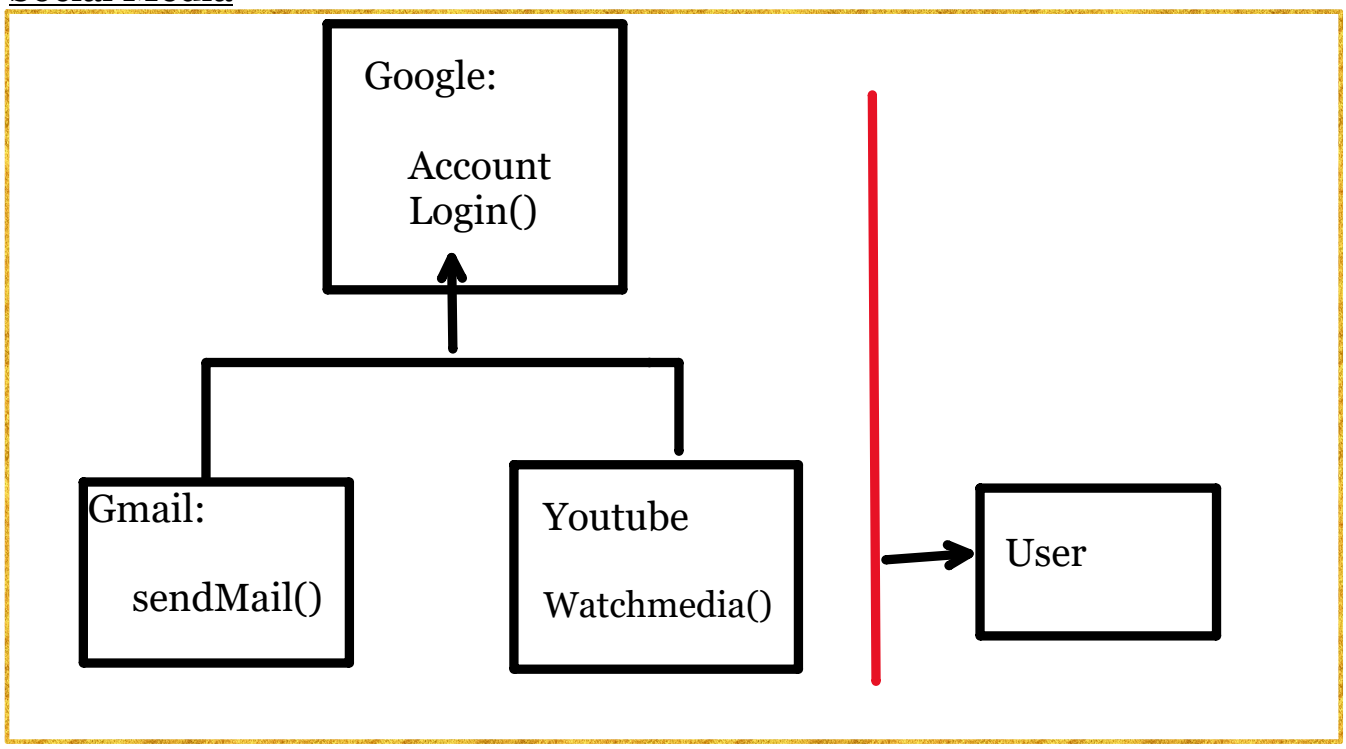
3. Hierarchical Inheritance:

Here, We have one parent class where the data is specified, but to this one parent we, will have N number of children classes.



```
class Mammals
{
   String type="Living beings";
}
class humans extends Mammals
{
}
class Dog extends Mammals
{
}
class Elephant extends Mammals
{
}
class Discovery//driver
{
   public static void main(String[] args)
   {
      System.out.println("Humans   :"+ new humans().type);
      System.out.println("Dogs     :"+ new Dog().type);
      System.out.println("Elephant :"+ new Elephant().type );
   }
}
```

Social Media



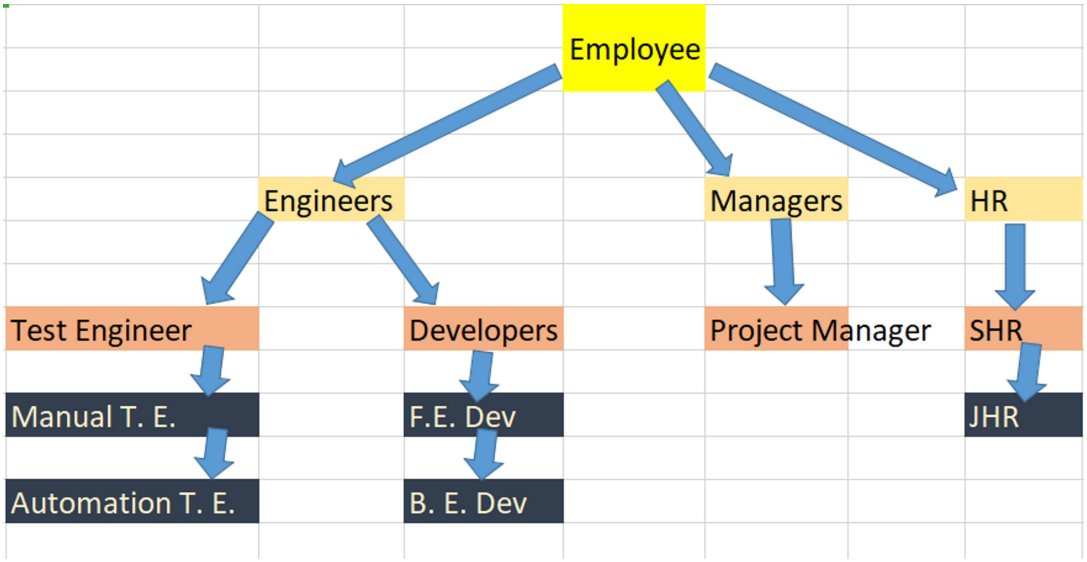

```
class Google
{
   String username,password;

   public Google(){}

   public Google(String username, String password)
```

```
    {
        this.password=password;
        this.username=username;
    }

    {
        System.out.print("Google Account created for :");
    }

    public boolean login(String u, String p)
    {
        if( u==username && p==password)
        {
            System.out.println("Login Successfull");
            return true;
        }
        else
        {
            System.out.println("Invalid credentials");
            return false;
        }
    }
}

class Gmail extends Google
{
    public Gmail(){}

    public Gmail(String username, String password)
    {
        super(username, password);
    }

    {
        System.out.println(" Gmail");
    }

    public void Emailing()
    {
        System.out.println(username+" is able to send or recieve email");
    }
}


class Youtube extends Google
{
    public Youtube(){}

    public Youtube(String username, String password)
    {
        super(username, password);
    }

    {
        System.out.println(" Youtube");
    }

    public void watch()
    {
        System.out.println(username+" can watch media content");
    }
}

class Mobile
{
    public static void main(String[] args)
    {
        Gmail g=new Gmail("Qsp","qwerty");
```

```
        Youtube y=new Youtube(g.username, g.password);

        if(y.login("Qsp", "qwerty"))
        {
           g.Emailing();
           y.watch();
        }
     }
}
```

4. Hybrid inheritance

This is a combination of two or more types of inheritance.



Suppose we have a program in a company where, there are categories for employees and here we are counting the total number of employees irrespective of the categories.

```
class Employee// Super
{
   static int ecount=0;

   public Employee(){}

   String name;
   int id;

   public Employee (String name, int id)
   {
      ecount++;
      this.name=name;
      this.id=id;
   }

   public void work()
   {
      System.out.println("Employee is working");
   }
}

class Manager extends Employee
{
   public Manager(){}
```

```
  public Manager(String name, int id)
  {
    super(name,id);
  }
  public void work()
  {
    System.out.println("Manager is managing the team");
  }
}

class ProjectManager extends Manager
{
  public ProjectManager(){}

  public ProjectManager(String name, int id)
  {
    super(name,id);
  }

  public void work()
  {
    System.out.println("Project manager is handling the project");
  }
}

class Developer extends Employee
{
 public Developer() {}

 public Developer(String name, int id)
 {
  super(name, id);
 }
 public void work()
 {
  System.out.println("Developer:"+name+" is Coding");
 }
}
class BEDeveloper extends Developer
{
 public BEDeveloper() {}

 public BEDeveloper(String name, int id)
 {
  super(name, id);
 }
 public void work()
 {
  System.out.println("B. E. Developer:"+name+" is writing the program");
 }
}
class FEDeveloper extends Developer
{

 public FEDeveloper() {}

 public FEDeveloper(String name, int id)
 {
  super(name, id);
 }
 public void work()
 {
  System.out.println("F. E. Developer:"+name+" is Designing");
 }
}
class TestEngineer extends Employee
{
  public TestEngineer() {}
```

```
  public TestEngineer(String name, int id)
  {
   super(name, id);
  }
 public void work()
 {   System.out.println("Test Engineer :"+name+" is Testing the code");
 }
}

class Company//driver
{
  public static void main(String[] args)
  {
    Employee e1=new Employee("abcd",1);
    e1.work();
    TestEngineer t1=new TestEngineer("Def",2);
    t1.work();
    Developer d1=new Developer("Ghi",3);
    d1.work();
    FEDeveloper f1=new FEDeveloper( "jkl", 4);
    f1.work();
    BEDeveloper b1=new BEDeveloper("abcde", 5);
    b1.work();
    ProjectManager p1=new ProjectManager("xyz",9999);
    p1.work();
    System.out.println("Employee Count is : "+e1.ecount);
  }
}
```

5. Multiple Inheritance

Here, one single child class can extend multiple parent which are at the same level.

When we inherit multiple parent classes, there might be a situation in which multiple parents can have same named data. Here, we get a confusion called as *"Ambiguity constraint".*

Because of this Multiple inheritance in java classes, is not possible.

This is also called as Diamond Problem in java

obj