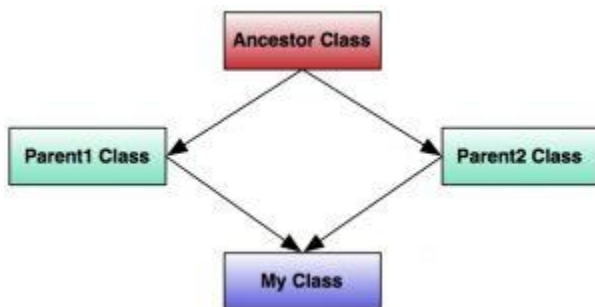


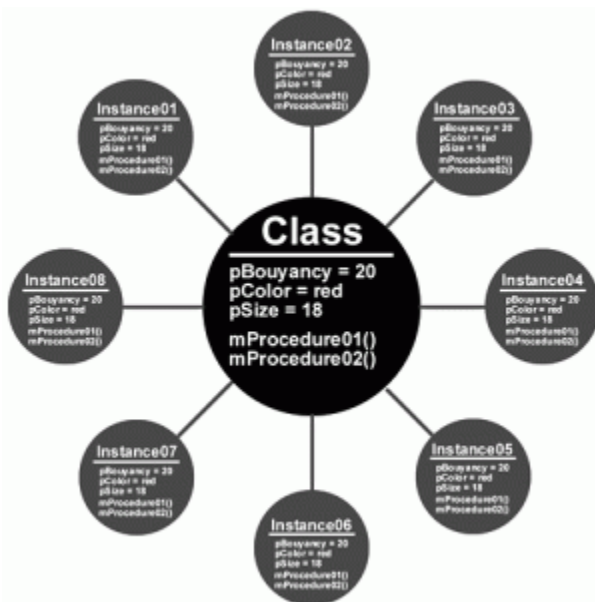
Classes & Objects

කොහොමද යාලුවනේ. අපේ thesigma blog අඩවියට පැමිණි ඔබ සෑම සාදරයෙන් පිළිගන්නවා. මෙහිදී අපි කතාකරන්නට බලාපොරොත්තු වෙනවා මොකද්ද මේ object oriented programming කියන්නේ කියලා. ගොඩක් ළමයින්ට නියෙන ජ්‍යෙෂ්ඨයන් නමයි object oriented programming පිළිබඳව හරි හැටි දැනුමක් නොමැතිකම. මේ හේතුව නිසා මම මගේ පළමු blog පෝස්ට් එක object oriented programming ගැන සිංහලෙන්ම ලියන්නට හිතුවා. මේ සඳහා අපි යොදාගැනීමට බලාපොරොත්තු වෙනවා java පරිගණක භාෂාව. මම හිතනවා ඔයාලට මෙයින් object oriented programming ගැන හොඳ අදහසක් ගන්නට පුළුවන් වෙයි කියලා. ඇත්තටම මේ object oriented කියන්නේ සංකල්පයක් ඉංග්‍රීසියෙන් කියනවනං concept එකක්. මෙයින් අපිට පුළුවන් අපේ programming සම්බන්ධ වැඩ පහසු කර ගන්නට. programming පැත්තෙන් හිතන්නම් මේ සංකල්පයෙහි සරල තේරුම නමයි සම්පූර්ණ ජ්‍යෙෂ්ඨයන් කොටස් කර coding කරලා එකට සම්බන්ධ කිරීමයි. ඉස්සර සම්පූර්ණ program එකම තනි පිටුවකට තමයි code කෙරේ. මෙහිදී විශාල program වල දෝෂ ආවම ඒවා නිවරදි කරන්නට බොහෝ කාලයක් මෙන්ම විශාල වෙහෙසක් දරන්නටත් සිදුවුණා. මෙන්න මේ හේතුව නිසා programmers ලා නව සංකල්පයකට යොමුවන්නට පෙළඹුණා. පසුව ඔවුන් තීරණය කළා මුළු program එකම අනු කොටස් වලට වෙන්කර code කර එකිනෙකට සම්බන්ධ කරන්නට. මෙන්න මේ සංකල්පයට නමයි object oriented programming කියලා කියන්නේ.

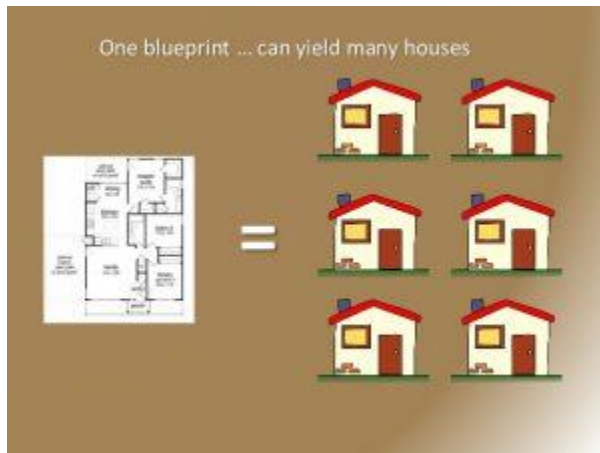


හරි එහෙනම් අපි බලමු මොකක්ද මේ object oriented programming කියන්නේ, මොකක්ද මෙහි විශේෂත්වය සහ කොහොමද මෙම සංකල්පය අපි code කරන විට යොදාගන්නෙ කියලා .

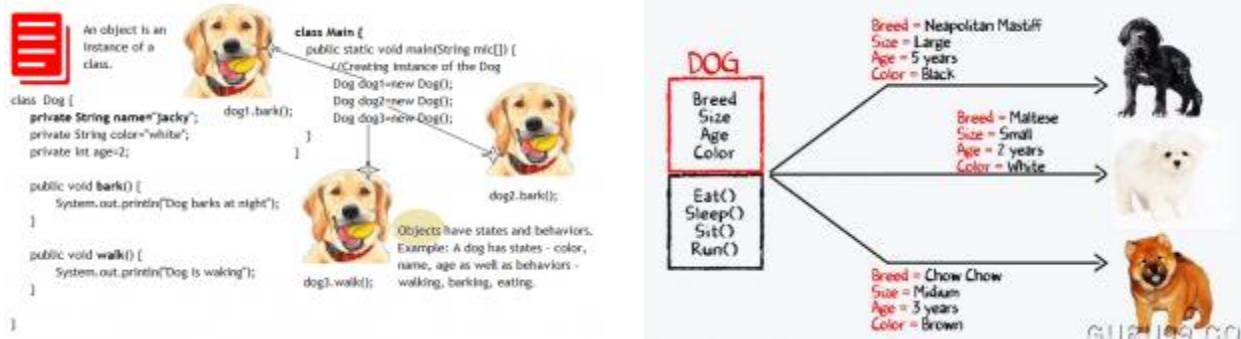
අන්තර්ගතය මේ සංකල්පය classes සහ object මත පදනම් වූවකි. අපි පළමුව බලමු මොකද්ද මේ class එකක් සහ object එකක් කියන්නේ කියලා. අන්තර්ගතය class එකක් කියලා කියන්නේ මොකක් හරි දෙයක් හදන්නට ගන්න template එකක් වගේ එකක් .උදාහරණයක් විදිහට ගන්නවනම් ගෙයක් හදන්නට ගන්න ජලයන් එක වගේ එකක්. එකමන් ජලයන් එක පාවිච්චි කරලා අපිට පුළුවන් ගෙවල් කිහිපයක් හදාරන්නට. මෙන්න මේ ජලයන් එකට නැත්නම් template එකට අපි පරිගණක භාෂාවේ කියනවා class එකක් කියලා. මෙන්න මේ ක්ලාස් එකෙන් හැදෙන වස්තුවට එහෙම නැත්නම් හැදෙන දේට අපි කියනවා object එකක් කියලා.



අපි කලින් උදාහරණයේ විදිහට නම් ජලයන් එක පාවිච්චි කරනවා හදපු ගෙදරක් වගේ. මේ එක ජලයන් එක පාවිච්චි කරලා විශාලත්වයෙන් අඩු හෝ වැඩි,වර්ණ වෙනස් වන ආකාරයට ගෙවල් කිහිපයක් සාදා ගන්න. එකම ජලයන් එකට අනුව සාදන නිසා මේ සෑම ගෙදරකම විශුභය එකම බව ඔබට තේරෙනවා ඇති .



පරිගණක භාෂාවට අනුව කියනවනම් එක class එකක් පාවිච්චි කරලා අපිට පුළුවන් object ගොඩාක් හදාගන්න . මේ විදිහට එකම ක්ලාස් එක පාවිච්චි කරල හදන හැම object එක විශ්වය එකමයි කියලා ඔයාලට තේරෙනවා ඇති . ඔන්න ඔය සරල සංසිද්ධිය object oriented කියන සංකල්පයේ නියත ප්‍රධානම දෙයක්. class එකක් ගන්නොත් properties දෙකක් තියෙනවා attributes සහ methods කියලා. attributes එකක් කියන්නේ class එකක් සතු දෙයක් .එහෙමත් නැත්නම් class එකක් පාවිච්චි කරලා අපි හදන object එකේ තියෙන දෙයක්.මෙන්න මේ attributes වලට අපි instance variables කියලත් කියනවා. උදාහරණයක් විදිහට අපි ගන්නොත් බල්ලා කියන class එක ගන්නොත් එකේ attributes වෙන්නේ පොදුවේ class එක සතු දේවල්ය.එනම් බල්ලා කියන class එකේ attributes විදිහට අපිට ගන්න පුළුවන් පාට, කකුල් ගණන වැනි දේවල්. methods එකක් කියන්නේ අපි හදන class එකේ හැසිරීම් ප්‍රකාශ කරන එකක්. අපේ බල්ලා කියන class එකේ නම් methods විදිහට අපිට ගන්න පුළුවන් දිවීම, බේරීම වැනි දේවල්.



මම හිතනවා classes සහ objects පිළිබඳ හොඳ අදහසක් ලැබෙන්න ඇති කියලා. ඊළඟ සිටින්න *thesigma blog* අඩවිය සමගින් මිලඟ object oriented පාඩම සමගින් හමුවෙමු. ස්තූතියි!

Access modifiers

ආයුබෝවන් ! object oriented programming දෙවෙනි පාඩමට ඔබ සැම සාදරයෙන් පිළිගන්නවා. කලින් පාඩමේදී අපි සාකච්ඡා කළා මොකද්ද object oriented concept එක කියන්නේ, ඇයි මේ සංකල්පය programming වලට ආවේ කියන එක , මොකක්ද class එකක් සහ object එකක් කියන්නේ කියලා සහ මොනවද classes වල හා object වල විශේෂත්වය කියලා. ඔයාලට කලින් පාඩම මගහැරුනා නම් අපේ blog අඩවියෙන් එක බැලුවොත් හොඳයි කියලා හිතනවා. හරි අපි දෙවැනි පාඩමට යන්න ඉස්සරින් java පරිගණක භාෂාව පාවිච්චි කරලා අපි class එකක් සහ ඒ class එක පාවිච්චි කරලා object කීපයක් සාදා ගනිමු . උදාහරණයක් විදිහට අපි ගනිමු dog class එක . මුලින්ම අපි dog class එකේ තියෙන attributes සහ methods හඳුනා ගනිමු. කලින් පාඩමේදී විදිහට අපිට ගන්න පුළුවන් attributes ලෙස අපිට ගන්න පුළුවන් color, age, breed වගේ දේවල්. methods විදියට ගන්න පුළුවන් run, sleep, eat වගේ දේවල් . හරි එහෙනම් අපි මේ හඳුනාගන්න attributes සහ method යොදාගෙන dog crush ක්ලාස් එක හදමු.

```
class Dog{

String color;
```

```

String breed;

int age;

public int run(int speed){

return speed; }

public void sleep(int time){

System.out.println("it every day sleep "+ time +" min"); }

public void eat(string food){

System.out.println("it eat "+ food); }

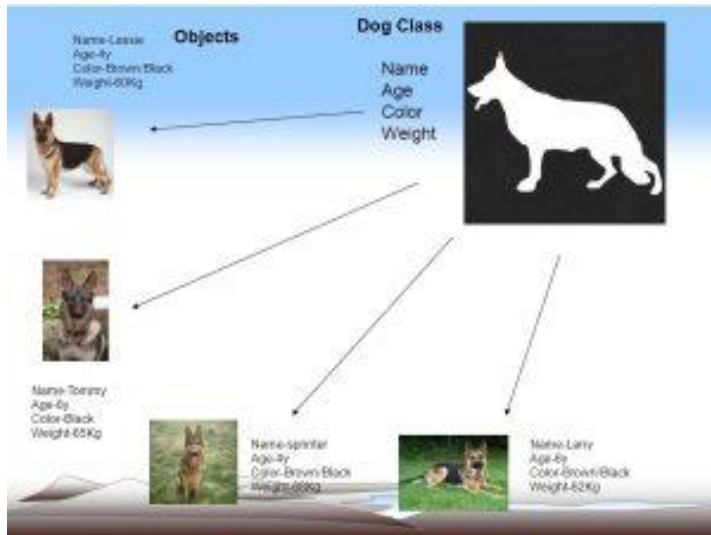
}

```

ඔබේ ඔය විදියට අපිට dog class එක හදාගන්න පුළුවන් .ඔබට java පිළිබඳව යම් දැනුමක් ඇත්නම් මේ නියෙන keywords තේරුම් ගැනීමට අපහසුවක් නැතුව ඇති කියලා හිතනවා. හරි අපි දැන් බලමු හදාගන්න class එක පාවිච්චි කරලා object කිහිපයක් සාදාගන්නට .ඒ සඳහා java පරිගණක භාෂාවේ දී පාවිච්චි කරනවා new කියන keyword එක.

```
Dog dog 1 = new Dog();
```

```
Dog dog 2 = new Dog() ;
```



හරි ඔන් න් ඔය විදියට තමයි class එකකින් object එකක් හදාගන්නේ .අපි මේ හදාගන්න object එකට values assign කරන්න ඕනෙ නම් 'dot' operator එක ජාවිච්චි කරලා මෙන්න මෙහෙමයි කරන්නේ .

```
dog 1.color = "Black" ;
```

```
dog 1. breed = "Chow Chow" ;
```

```
dog 1. age = 2 ;
```

ඔන් න් ඔහොම තමයි object එකක නියන attributes වලට values assign කරන්නේ. class එකේ methods නියෙනවනම් ඒවා මෙන්න මේ විදිහට call කරනවා.

```
dog 1.run ( 10 );
```

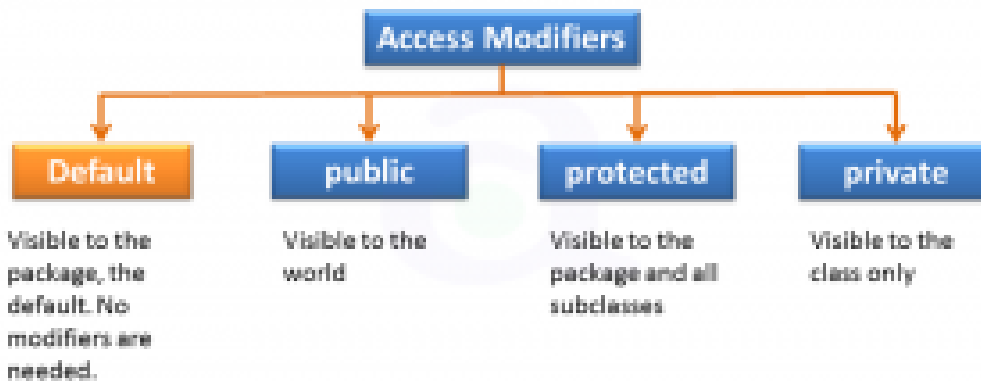


```
dog 1.sleep ( 100 );
```

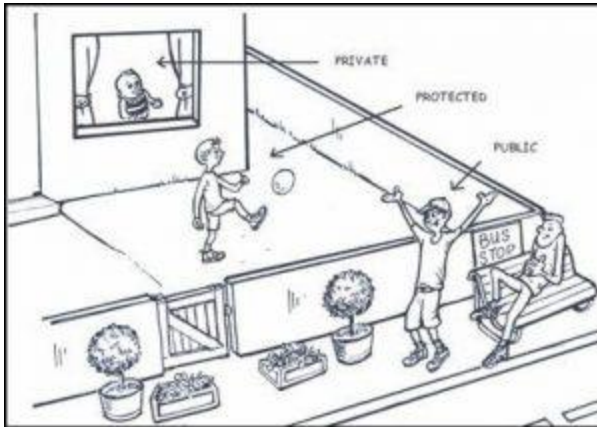
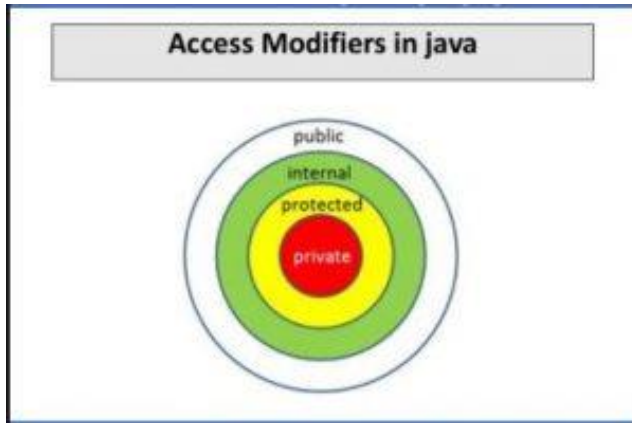
```
dog 1.eat ( "meat");
```

ඔබ්බා ඔය විදිහට තමයි class එකක් හදලා එකේ attributes වලට values assign කරන්නෙත් methods call කරන්නෙත්. මේ විදිහට එක class එකකින් අපිට පුළුවන් විවිධ values assign කරලා object ගොඩක් හදාගන්න.

හරි අපි දැන් බැලුවා object oriented concept එක පාවිච්චි කරලා කොහොමද class එකක් සහ object එකක් හදාගන්නෙ කියලා. දැන් අපි බලන්න යන්නේ object oriented concept එකේ නියත නවත් පොඩි කොටසක් සම්බන්ධව . ඒ තමයි members නැත්නම් access modifiers ගැන. මොකක්ද මේ access modifiers කියලා කියන්නේ? ඇත්තටම access modifiers කියන්නේ ඔයාගේ ක්ලාස් එකට කාටද ඔයා access දෙන්නේ කියන එක. තවත් විදියකින් කියනවා නම් ඔයාඔයාගේ class එකට කාටද ඇතුළු වෙන්න පුළුවන් කියන එක . ඇත්තට මේ access modifiers වලින් කරන්නේ ඔයාගේ class එක බාහිර පුද්ගලයන්ගෙන් ආරක්ෂා කරන ආරක්ෂා කරන එක. class එක ඇතුළෙන් නියත methods සහ attributes වල ආරක්ෂාවටත් මේ access modifiers පාවිච්චි කරන්න පුළුවන්. access modifiers හතරක් නියතවා public , private , protected සහ default කියලා.



public access modifier එකෙන් පුළුවන් ඕනෑම බාහිර පුද්ගලයෙකුට ඇතුළුවන්න. අපි කිසිම access modifier එක දාගන්නේ නැතුව class එකක්, attributes හෝ methods සකස් කළොත් ඒකෙන් කියවෙන්නේ default access modifier එක ගැන. default access modifier එකෙන් තමන්ගේ package සිටින සියලුම පුද්ගලයින්ට සහ තමන්ගේ class එකේ අයට පමණක් ඇතුළු වන්න පුළුවන්. protected access modifier එක පාවිච්චි කළොත් තමන්ගේ class එකේ අයටයි sub class වල අයටයි දෙගොල්ලටම ඇතුළු වන්න පුළුවන්. private access modifier එක පාවිච්චි කළොත් තමන්ගේ class එකේ අයට පමණක් ඇතුළුවන්න පුළුවන්.



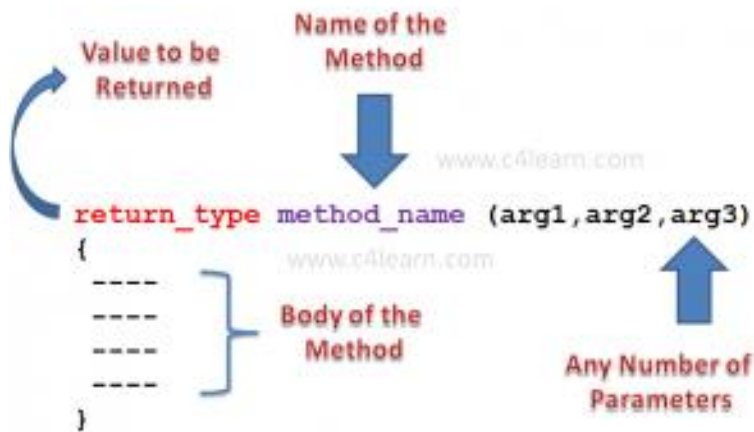
ඉහල තියෙන පින්තූර දෙකෙන් තවදුරටත් අදහසක් ගත හැකි access modifiers ගැන. අද කරපු පාඩම තේරුනා කියලා හිතනවා. අදට පාඩම ඉවරයි. මිලිග පාඩමෙන් හමුවෙමු. ස්තූතියි !

Method Overloading

කොහොමද යාලුවනේ ! ඔන්න අදත් මං ආවා oriented object oriented concept එක තියන අලුත්ම පාඩමක් සමගින්. පුරුදු විදිහටම object oriented පාඩම් මාලාවේ තෙවැනි පාඩමත් සමග එක්වන ඔයාලා ඔක්කොම සාදරයෙන් පිළිගන්නවා. කලින් පාඩම් සමග ඔයාලා සම්බන්ධ වුණා නම් දැන් දන්නවා ඇති මොකක්ද object oriented concept කියන්නේ, class එකක් සහ object එකක් අපි හදා ගන්නේ කොහොමද කියලා සහ access modifiers කියන්නේ මොකක්ද කියලා. තුන්වෙනි පාඩම පටන් ගන්න ඉස්සරින් access modifiers ගැන පොඩ්ඩක් මතක් කරන්නම්. ඇත්තටම access modifier එකකින් කරන්නේ තමන්ගේ class එකට සහ class එක ඇතුලේ තියන attributes වලට හා methods වලට ආරක්ෂාව සපයන එක.

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

ඔත්ත ඕක තමයි access modifiers ගැන නියන සාරාංශය. access modifiers ගැන ඔයාලට අමතක වෙලා තිබුන නම් දැන් මතක් වෙන්න ඇති කියලා හිතනවා. හරි එහෙනම් අපි බලමු අද පාඨම මොකක්ද කියලා. අද අපි වැඩිපුර methods ගැන කතා තමයි කරන්න බලාපොරොත්තු වෙන්නේ. ඔයාලා දැනටමත් දන්නවා මොකක්ද method එකක් කියන්නේ කියලා . තවදුරටත් අපි methods ගැන කියනවනම්, methods එකක් නම් ඒකට return type එකක් තියෙන්න ඕන. return type එක කියන්නේ methods එකෙන් අපට මොන වර්ගයේ value එකක් ද එළියට ගන්න ඕනි කියන එක. උදාහරණයක් විදිහට ගන්නවනම් අපට යම්කිසි method එකක් පාවිච්චි කරලා යම්කිසි ඉලක්කමක් method එකෙන් එළියට ගන්න ඕනි නම් අපට පාවිච්චි කරන්න පුළුවන් int, double හෝ float වගේ data type එකක්. මොකක් හරි වචනයක් අපට මෙනත දෙකෙන් එළියට ගන්න ඕනි නම් අපට පාවිච්චි කරන්න පුළුවන් string කියන data type එක. කිසිම value එකක් අපට method එකෙන් එළියට ගන්න ඕනි නැතිනම් අපට void කියන return type එක පාවිච්චි කරන්න පුළුවන්. ඔත්ත ඕකට තමයි return type එක කියලා කියන්නේ. මීට අමතරව method එකක් නම් ඒකට අපි කලින් කතා කරපු access modifier එකක් තනියෙන්ම ඕනි. අපි කලින් කලා වගේ method එකකට parameter එකක් හෝ parameters කිහිපයක් assign කරලා අපට method එක ඇතුලට values pass කරන්නත් පුළුවන්.



ඔබ්බ ඔය විදිහට නමයි method එකක් හදන්නේ. ඔබ්බ ඔය method එකේ නමත් එක්ක වරහන් ඇතුලේ නියන parameters සෙට් එකට අපි කියනවා signature එක කියලා return type එක signature විදිහට අපි ගන්නෙ නෑ.

දැන් අපි කතා කරන්න යන්නේ දැන් අපි කතා කරන්න යන්නේ methods වල නියෙන විශේෂ අවස්ථා දෙකක් ගැන. ඔයා හිතන්න ඔයා ඔයාගේ class එකේ නමින් methods දෙකක් හෝ කිහිපයක් නියෙනවා කියලා. ඔයා දැන් object එකක් හදලා එක method එකක් call කළොත් compiler එකට හිතාගන්න බැරිවෙනවා කොයි method එකද call කරන්නේ කියලා. අන්න එතකොට ඔයාගේ program එකේ Error එකක් පෙන්නනවා. අන්න ඒක අපිට වලක්වා ගන්න පුළුවන් ඔයාගේ method එකේ වරහන් ඇතුලේ නියෙන parameters ගණන හරි ඒ වගේ data type එක හරි වෙනස් කරලා .පසුව ඔයාට පුළුවන් ඔයාට අවශ්‍ය method එක call කරවන්න. ඔබ්බ ඔකට අපි කියනවා method overloading කියලා. method overloading කියන්නේ object oriented concept එක නියන ඉතා විශේෂ දෙයක්. method overloading ටිකක් තේරෙනවා මදි වගෙ නම් අපි උදා උදාහරණයක් බලමු.

හරි මම හදනවා class එකක් හැඩතල වල වර්ගඵලය සොයන්න. හැඩතල විදිහට මම ගන්නවා සමචතුරස්රය වෘත්තය හා සෘජුකෝණාස්රය. සමචතුරස්රයක වර්ගඵලය සොයන්න නම් එකට value එකක් හොඳටෝම ඇති. වෘත්තයකත් එහෙමමයි. සෘජුකෝණාස්රයක නම් දිග හා පළල කියලා values දෙකක් දෙන්න වෙනවා. මම හදන class එකේ method එකේ නම විදිහට මම ගන්නවා area කියලා. method overloading නිසා හැඩතල සියල්ලගෙම වර්ගඵලය සෙවීමට දන්නා method වල නම් area කියන එකම විය යුතුයි. දැන් අපිට මේ method එකිනෙකින් වෙන්කර ගැනීමට නම්

parameter signature එක වෙනස් කරන්න වෙනවා කියලා ඔයාලට තේරෙනවා ඇති. සෘජුකෝණාස්‍රයක වර්ගඵලය සෙවීමට parameters දෙකක් අවශ්‍ය නිසා ඒ method එක අනික් method දෙකෙන් වෙන්කර ගන්න පුළුවන්. අනික් method දෙක එකිනෙකින් වෙන්කර ගැනීමට මම වෘත්තයක වර්ගඵලය සොයන method එකේ parameter එකේ data type එක float ලෙසද අනෙක සඳහා integer අගයක්ද ලබා දෙනවා. දැන් ඔබට තේරෙනවා ඇති අනෙක් method දෙකෙන් parameter signature එක වෙනස් කියලා. ඔන්න ඔය විදිහට එකම නම තියන method වල parameters signature එක වෙනස් කරලා තමයි අපි method overloading කරන්නේ. හරි එහෙනම් දැන් අපි අපේ වර්ගඵලය සොයන class එක හදමු.

```
class Area{

    public void area(float x){

        float c=3.14*x*x;

        System.out.print("aria of circle is :"+c);}

    public void area(int x){

        int A=x*x;

        System.out.print("aria of square is:"+A);}

    public void area(int x,int y){

        int A=x*y;

        System.out.print("aria of rectangle is :"+A);}
```

```
}
```

```
class CalArea{
```

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

```
        Area A1=new Area();
```

```
        // if you want to calculate area of circle,pass a float value as an argument..
```

```
        A1.area(5.2);
```

```
        // if you want to calculate area of square ,pass an integer value as an argument..
```

```
        A1.area(5);
```

```
        // if you want to calculate area of rectangle, pass two integers as the arguments..
```

```
        A1.area(5,10);
```

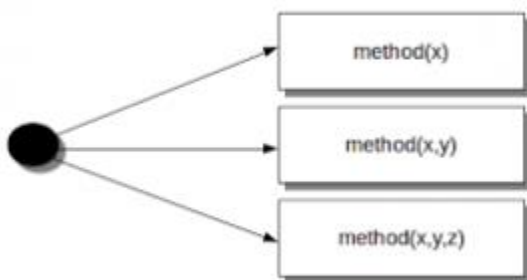
```
    }
```

}

ඔබ්බා ඔබ්බාම නමයි method overloading කරන්නේ. අද කරපු පාඩම තේරුනා කියලා හිතනවා. method overloading එපමණයි. method overriding සහ constructor එක ගැන මිලඟ පාඩමෙන් කියලා දෙන්නම්.ස්තූතියි !

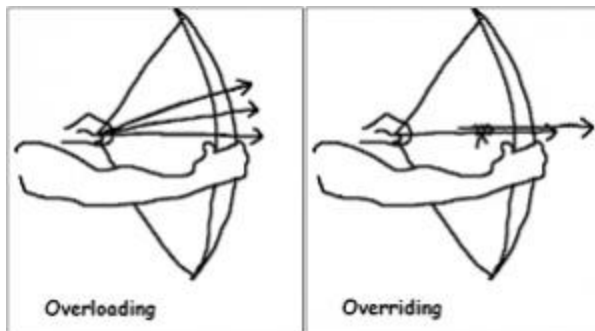
Constructor Method

ආයුබෝවන්! ඔබ්බා මං අදත් ආවා කලින් පාඩමේ ඉතිරි කොටසත් සමගින්.object oriented programming සිව්වන පාඩමට ඔයාලා ඔක්කොම සාදරයෙන් පිලිගන්නවා. කලින් පාඩමේදී අපි කතා කලා method overloading ගැන.පොඩි මනක් කිරීමක් කරමු method overloading ගැන, ඇත්තටම method overloading කියන්නේ එකම නමින් method කිහිපයක් declare කිරීම ඒවගෙ parameter signature එක වෙනස් කරලා.parameter signature එක වෙනස් කරන නිසා compiler එකට පුළුවන් ඒ එක එක methods වෙන වෙනම එකිනෙකින් වෙන් කරලා හදුනා ගන්න.ඔබ්බා ඔබ්බා අපි method overloading කියලා කිව්වෙ.කෙටියෙන්ම කියනවනම් මෙන්න මේකයි



හරි method overloading ගැන හොඳටම මනක් උනා කියලා හිතනවා.එහෙනම් අපි බලමු අද පාඩම මොකක්ද කියලා.අද අපි සලකා බලන්න යන්නෙ method overriding ගැන.method overriding කියලා කියන්නෙ කෙටියෙන්ම කව්වොත් method එකක body එක අපිට ඕනි විදියට වෙනස් කරලා පාවිච්චි කරන එක.method overriding වැඩිපුරම පාවිච්චි කරන්නෙ object oriented concept එකේ නියෙන inheritance කියන කොටසේ.method

overriding කියන එක ගැන හරි හැටි තේරුනේ නෑ නේද? කමක් නෑ අපි inheritance කියන කොටසේදී method overriding ගැන වැඩි දුරටත් කතා කරමු. ඇත්තටම method overloading එකේදී කරන්නේ එකම නමින් methods ගොඩක් හදන එකයි. නමුත් method overriding එකේදී කරන්නේ එක method එකක් අරගෙන එකේ body එක වෙනස් කරන එකයි. උදාහරණයක් විඩියට කිව්වොත් මෙන්න මේ වගේ.



ඔන්න ඔකයි method overloading හා method overriding අතර නියෙන ප්‍රධානම වෙනස්කම. හරි method overriding ගැන පසුවට කතා කරන්නට ඉතුරු කරලා අපි දැන් බලමු constructor එක කියන්නේ මොකද්ද කියලා. object oriented concept වල නියෙන ඉතාම වැදගත් කොටසක් නමයි constructor එක කියන්නේ. constructor එක කියන්නේ මොකද්ද කියලා කෙනෙක් ඇහුවොත් දෙන්න නියෙන සරලම උත්තරේ නමයි, constructor එක කියන්නේ method එකක්. නමුත් ටිකක් විශේෂ method එකක්. constructor එකේ විශේෂත්ව කීපයක්ම තියනවා. constructor එකක් කියන්නේ method එකක් කියලා කලින් මම කිව්වනේ. ඒ method එකේ නැත්නම් constructor එකේ නම විදියට අපි ගන්නේ class එකේ නමමයි. ඔන්න ඔක constructor එකක නියෙන ප්‍රධාන විශේෂත්වයක්. මෙන්න මේ හේතුව නිසා class එක පාවිච්චි කරලා object එකක් හදන සෑම විටම constructor method එක call වෙනවා. උදාහරණයක් විදියට car class එක සලකමු. ඒකේ method එකක් ලියනවා මම drive කියලා. class එකේ නමින්ම method එකකුත් හදනවා. වෙනත් විදියකින් කියනවනම් constructor එකක් හදනවා. අත්තටම කිව්වොත් constructor එක හදන්න දෙයක් නැහැ. අපි object එකක් හදන විට constructor එකක් auto හැදිලි call වෙනවා අපිට ජේන්නේ නතුවට. ඔන්න ඔක constructor එකක නියෙන ප්‍රධාන විශේෂත්වයක්. හරි එහෙනම් අපි car class එක හදමු.

```
class Car{
```

```
Car(){  
  
    System.out.println("this is my constructor ");  
  
}  
  
void drive(){  
  
    System.out.println("I can drive fast");  
  
}  
  
}  
  
class MyCar{  
  
    public static void main(String a[]){  
  
  
        Car car1=new Car();  
  
    }  
  
}
```

අපි බලමු දැන් මෙහි out put එක මොකක්ද කියලා.

```
Command Prompt
C:\Users\USER PC\Desktop>javac MyCar.java
C:\Users\USER PC\Desktop>java MyCar
this is my constructor
C:\Users\USER PC\Desktop>
```

හරි දැන් ඔයාලට තේරෙනවා අති constructor එක කියන්නේ මොන වගේ එකක්ද කියලා. අපි object එකක් හැදුවාම constructor එක auto call වෙනවා. හරි මම ඔයාලට කලින් කීවා constructor එක කියන්නේ method එකක් කියලා. method එකක් නම් return type එකක් තියෙන්න ඕනෙනෙ. නමුත් මගේ method එකේ නැත්නම් constructor එකේ return type එකක් නැතේ. ඔන්න ඕක තමයි constructor එකක තියෙන අනෙක් විශේෂය. ඒ කියන්නේ constructor එකකට return type එකක් නැහැ. ඒ කියන්නේ කිසිමදෙයක් return කරලා එලියට ගන්න බැහැ. කෙටියෙන්ම කීව්වොත් constructor එකක් කියන්නෙ return type එකක් නැති, object එකක් හදන වෙලාවෙ පමනක් call වෙන, class එකේ නමින්ම තියන method එකක්. අනික් methods වගෙම constructor එකත් overload කරන්න පුළුවන්. එ ගැන අපි මීලග පඩමේදි කතා කරමු. අදට පඩම අවසන්. පඩම් අංක පහෙන් හමුවෙමු. ස්තූතියි!

Constructor Overloading

ඔන්න යාලුවේ අදත් මං ආවා කලින් පාඩමේ ඉතිරි ටිකත් සමගින්. කලින් පාඩමේදි අපි කථා කලා method overriding සහ constructor එක ගැන. මොකද්ද අපි method overriding කියල කීව්වෙ? method overriding. කියල අපි කීව්වෙ එකම නමින් තියෙන method එකක body එක අපිට ඕනි විදියට වෙනස් කිරීමයි. method overloading හා method overriding අතර වෙනසත් අපි කතා කලා. කලින් පාඩම අමතකනම් හරි බැලුවෙ නැත්නම් හරි බැලුවොත් හොඳයි කියල හිතනවා. method overriding වලට පසුව අපි කතා කලා constructor එක ගැන. මොකද්ද අපි constructor එක කියල කීව්වේ?, constructor එක කියල අපි කීව්වෙ, class එකේ නමින්ම තියෙන object

එකක් හදන වෙලාවෙදි පමණක් call වෙන return type එකක් නැති method එකකට.හරි දැන් constructor එක ගැනත් මතක් වෙන්න ඇති කියල හිතනවා.

හරි අද පාඩමේදි අපි කතා කරන්න යන්නේ constructor එක හරහා values pass කරන්නේ කොහොමද කියන එක ගැන. constructor එක කියන්නේ method එකක්මයි කියල මං කලින් පාඩමේදි කිව්වනෙ.එහෙනම් method එකක values pass කරන විදියටම අපිට පුලුවන් constructor එක හරහාත් values pass කරන්න.හැබැයි ඊට කලින් attributes හා ඒවාට අදාල parameters ටික constructor එකේ වරහන් ඇතුලෙ define කරගෙන ඉන්න ඕනි කලින්ම.පසුව constructor එකේ body එක ඇතුලෙ attributes වලට ඔයා ගන්න parameters ටික assign කර ගන්නත් ඕනි.ඔන්න ඔය ටික තමයි constructor එකක් හරහා values pass කිරීමේදි ඉස්සරෝම කරන්න ඕනි.method එකක නම් object එකක් හදාගෙන dot operator එක පාවිච්චි කරල method එක call කරලානෙ values pass කරන්නෙ.constructor එකේ අපි values pass කරන්නෙ object එක හදන වෙලාවෙමයි.ටිකක් තේරුනා මදි වගේ නේද? එහෙනම් මෙන්න මේ උදාහරණය බලන්නකෝ.

මම හදනව class එකක් car කියල.brand කියලා attribute එකකුත් හදා ගන්නවා class එක ඇතුලෙම.එතකොට car class එකේ constructor එක වෙන්නේ car() කියලනේ.හරි දැන් මට ඕනි car1 කරලා object එකක් හදලා car1 වල brand එක constructor එක හරහා value එකක් විදියට pass කරලා object එක හදනකොට car1 වල brand එක print කරන්න.මෙන්න මෙහෙම කලා නම් හරි නේද බලන්නකෝ.

```
class Car{

    Car(String brand){

        System.out.print("My Car brand is "+brand);

    }
```

```

}

class MyCar{

    public static void main(String a[]){

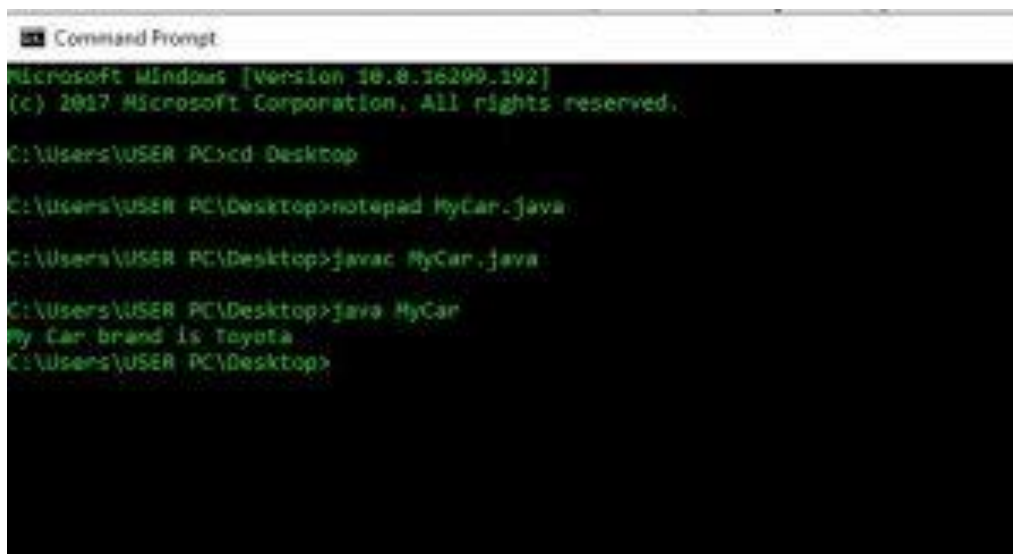
        Car car1 = new Car("Toyota");

    }

}

```

Output :



```

Command Prompt
Microsoft Windows [Version 10.0.16299.192]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\USER PC\Desktop>
C:\Users\USER PC\Desktop>notepad MyCar.java
C:\Users\USER PC\Desktop>javac MyCar.java
C:\Users\USER PC\Desktop>java MyCar
My Car brand is Toyota
C:\Users\USER PC\Desktop>

```

හරි දැන් ඔයාලට පෙනවා ඇති car1 කියන object එක හැදෑවම brand එක print වෙලා තියනවා කියන එක.ඒ කියන්නේ constructor එක හරහා value එකක් pass වෙලා තියෙන බව.ඔන්න ඔහොම තමයි constructor එක හරහා values pass කරන්නේ.හැමෝටම තේරුනා කියල හිතනවා. හරි අපි කලින් කීව්වනෙ constructor එක කියන්නේ method එකක් කියල.දැන් ඔයාලට ප්රශ්නයක් ඇති constructor එක overload කරන්න

පුලුවන්ද කියලා.කෙටියෙන්ම කිව්වොත් පුලුවන්.ඇයිසේ කියනවනම් constructor එක කියන්නෙන් method එකක් නිසා. අනෙක් method overloading කරන විදියටම නමයි constructor එකත් overloading කරන්නේ.එකම විශේෂත්වය නමයි method overloading එකේදි අපි කරන්නෙ object එක හදලා අදාල method එක call කරන එකනෙ.constructor එකක් overload කිරීමේදි අපි කරන්නේ object එක හදනකොටම values දීල.අපි ලබාදෙන values ගනනෙන් හරි ඒවායේ data type එකේ වෙනසින් හරි නමයි compiler එක හදුනා ගන්නේ කොයි constructor එකද මේ call කරන්නෙ කියල.ඔන්න ඔය constructor එක overload කරන එකට අපි කියනවා constructor overloading කියලා.තේරුනා මදි වගේනම් මෙන්න මේ උදාහරණයත් බලන්නකෝ.මම මගෙ කලින් පාඩමේ හඬනල වල වර්ගපලය හොයන උදාහරනයම ගන්නම්.

```
class Area{

    double c;

    int A;

    Area(double x){

        c=14*x*x;

        System.out.print("aria of circle is :"+c);}

    Area(int x){

        A=3.14*x*x;

        System.out.print("aria of square is:"+A);}
```



```
Area(int x,int y){  
  
    A=x*y;  
  
    System.out.print("aria of rectangle is :"+A);}  
  
}
```

```
class CalArea{
```

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

```
        // if you want to calculate the area of circle,give a double value in  
        creating an object
```

```
        Area A1=new Area(5.5);
```

```
        // if you want to calculate the area of square,give an integer value in  
        creating an object
```

```
        Area A2=new Area(5);
```

```
        // if you want to calculate the area of rectangle,give two integer values  
        in creating an object
```

```
        Area A3=new Area(5,10);
```

```
}
```

```
}
```

Output :

```
Command Prompt
C:\Users\US68\PC\Desktop>java CalArea.java
C:\Users\US68\PC\Desktop>java CalArea
Area of circle is :34.983
Area of square is :25
Area of rectangle is :36
C:\Users\US68\PC\Desktop>
```

හරි දැන් තේරුනා නේද constructor overloading කියන්නෙ මොකක්ද කියලා.හරි අපි දැන් බලන්න යන්නේ object oriented concept එකේ නියෙන වැදගත් keyword එකක් ගැන.ඒ නමයි 'this' කියන keyword එක.this keyword එකෙන් කරන්නෙ method එකෙන් එලියෙ නියෙන attribute එකකට call කරන එක. this keyword එකේ කාර්යය මොකක්ද කියලා මෙන්න මේ උදාහරණයෙන් තවදුරටත් පැහැදිලි කර ගනිමු.කලින් උදාහරණයම මං ගන්නවා.constructor එකේ parameter එක විදියට මම attribute එකේ නමම නියෙන parameter එකක් ගන්නවා .ඒ කියන්නෙ parameter එකේ නම විදියට brand කියන එකම ගන්නවා.දැන් constructor එක call කලොත් ඔයාලට ජේනව ඇති null කියලා නමයි print වෙලා නියෙන්නෙ.(මට අවශ්‍ය වෙන්නේ attribute එකේ value එක නිසා මම print කරනවා this.brand කියලා.)

```
Command Prompt
C:\Users\USER\PC\Desktop>javac MyCar.java
C:\Users\USER\PC\Desktop>java MyCar
My car brand is :null
C:\Users\USER\PC\Desktop>
```

ඒ කියන්නේ අපි දුන්න value එක හරි හැටි attribute එකට pass වෙලා නැහැ.මෙන්න මේ ගැටලුව විසදන්න අපිට මේ this කියන keyword එක භාවිත කරන්න පුලුවන්.ඔන්න බලන්නකෝ this keyword එක භාවිත කලාම output එක.

```
class Car{

    String brand;

    Car(String brand){

        this.brand = brand;

        System.out.print("My car brand is :"+this.brand);

    }

}
```

```

class MyCar{

    public static void main(String a[]){

        Car car1=new Car("toyota");

    }

}

```

Output :

```

C:\Users\USER PC\Desktop>javac MyCar.java
C:\Users\USER PC\Desktop>java MyCar
My car brand is :null
C:\Users\USER PC\Desktop>javac MyCar.java
C:\Users\USER PC\Desktop>java MyCar
My car brand is :toyota
C:\Users\USER PC\Desktop>

```

දැන් value එක හරි හැටි pass වෙලා තියනවා කියලා ඔයාලට තේරෙනව
 ඇති.this keyword එකෙන් වෙන්තෙ ඔන්න ඔය කාර්යය තමයි.ඒ කියන්නෙ
 method එකක තියෙන parameter එකක් හෝ කිහිපයක් class එකේ define

කරලා තියෙන attributes වල නමින්ම නියනවනම් ,ඒ ඒ attribute වලට ඊට අදාල parameter එකේ value එක assign කිරීමට මේ this කියන keyword එක පාවිච්චි කරන්න පුලුවන්.හරි දැන් තේරුනා නේද this keyword එක පාවිච්චි කරන්නෙ ඇයි කියලා.අද පාඩම මෙනෙත් නිමයි.මීලග පාඩම තුලින් හමුවෙමු.ස්තූතියි!

Inheritance

ආයුබෝවන්! කොහොමද යාලුවනේ ! object oriented මීලග පාඩමත් සමගින් ඔන්න මං අදත් ආවා ඔයාල හමුවෙන්න.object oriented මීලග කොටසට ඔයාලා හැමෝම සාදරයෙන් පිලිගන්නවා.කලින් පාඩමේදි අපි කලා කලා constructor overloading ගැන හා this කියන keyword එක ගැන.හැමෝටම මතකයි නේද? පොඩි හරි අමතක වීමක් නියනවනම් කලින් පාඩම ඉක්මනින් බලන් එන්නකෝ.අද නම් මම කලින් පාඩම මතක් කරන්න යන්නෙ නැහැ.ඇයියේ දන්නවද? ඒ තමයි අපි අද බලන්න යන්නෙ object oriented concept එකේ තියෙන අලුත්ම කොටසක් ගැන.ඒ කොටස පටන් ගන්න ඉස්සරින් ඒ ගැන පොඩි හැදින්වීමක් කරන්නම්.ඇත්තටම කියනවනම් අපි මේ ඉදිරියට කලා කරන්න යන කොටස් කිහිපය තමයි object oriented concept එකේ තියෙන වැදගත්ම ටික.ඒ නිසා මේ පාඩම් කිහිපයේ මොකක් හෝ පොඩි අපැහැදිලි නැතක් තිබුනොත් කොමෙන්ටුවක් දාන්න ඕනි හොඳේ☺☺

හරි එහෙනං අපි බලමු මොකක්ද අද පාඩම කියලා.මෙනැත් සිට ඉදිරියට අපි කලා කරන්න බලාපොරොත්තු වන්නේ object oriented concept එකේ තියෙන features කිහිපයක් ගැන.මේවා අපිට ප්රධාන වශයෙන් කොටස් හතරකට වෙන් කරන්න පුලුවන් මෙන්න මේ විදියට.

1. Inheritance

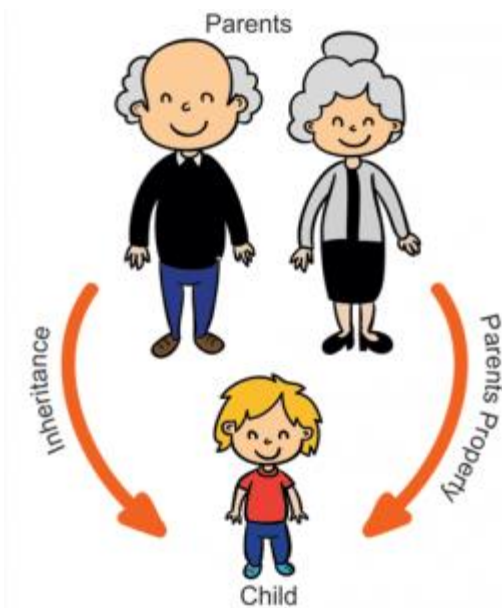
2.Encapsulation

3.polymorphism

4.Interface

කියලා.අද අපි කලා කරන්න යන්නෙ මෙහි පලමු වැන්න ගැන.ඒ කියන්නෙ inheritance ගැන.ඇත්තටම යාලුවනේ මොකක්ද මේ inheritance කියන්නේ.inherit කරනවා කියන එකේ සරලම අදහස තමයි එක

පරම්පරාවකින් තවත් පරම්පරාවකට යම් කිසි දෙයක් දායාද කරනවා කරනවා කියන එකයි. හරියට නිකන් මෙන්න මේ වගේ .



එහෙම කිව්වම ඔයාලා බලනවා ඇති කොහොමද මේක programming වලට සම්බන්ධ වෙන්නෙ කියලා. මෙන්න මෙහෙමයි inheritance කියන feature එක programming වලට අපි සම්බන්ධ කර ගන්නෙ.හරි මං උදාහරණයකින්ම පැහැදිලි කරන්නමිකෝ.

දැන් ඔයා හිතන්න ඔයාට කෙනෙක් කියනවා teacher,student හා doctor කියලා object තුනක් හදන්න කියලා.එතකොට ඔයා මොකද කරන්නෙ?ඉස්සරෝම class තුනක් හදනවා teacher , student හා doctor කියලා.ඊට පස්සෙ object තුන හදනවා.code එක ලිව්වොත් මෙන්න මේ වගේ නේද.

```
class Teacher{  
  
    String name;  
  
    int age;  
  
    String Gender;  
  
    String Designation;  
}
```



```
int salary;

void eat(){

    // statements

}

void walk(){

    // statements

}

void teach(){

    // statements

}

void takeExams(){

    // statements

}

}
```

```
class Doctor{

    String name;

    int age ;

    String Gender;

    String Designation;

    String salary;

    void eat(){

        // statements

    }

    void walk(){

        // statements

    }

    void ChackUp(){

        // statements

    }
```

```
void prescribe(){  
  
    // statements  
  
}  
  
}
```

```
class Student{  
  
    String name;  
  
    int age;  
  
    String Gender;  
  
    String progame;  
  
    int studyYear;  
  
    void eat(){  
  
        // statements  
  
    }  
  
    void walk(){
```

```
        // statements

    }

    void study(){

        // statements

    }

    void heldYear(){

        // statements

    }

}

// then you can create objects

class CraateObject{

    public static void main(String s[]){

        // creating teacher object
```

```

Teacher teacher1=new Teacher();

// creating doctor object

Doctor doctor1=new Doctor();

// creating student object

Student student1=new Student();

}

}

```

ඔයාලා කරපු ඒ ක්රමය හරියටම හරි.කිසිම වැරද්දක් නැ.හැබැයි මම කියන දේත් පොඩ්ඩක් අහන්නකෝ.පොඩ්ඩක් හොදට කල්පනාවෙන් ඇස් ඇරලා බලන්නකෝ ඔයා හදපු class තුනම දිහා.class තුනේම එක වගේ attributes සහ methods කීපයක්ම තියනවා නේද?තවත් විදියකින් කිව්වොත් class තුනටම පොදු attributes හා methods කිහිපයක්ම තියනවා .ඒ පොදු attributes හා methods ටික එකම class එකකට ගන්නොත් හොදයි කියලා හිතෙනවා නේද. හරි එහෙනත් අපි class තුනටම පොදු attributes හා methods ටික එකම class එකකට ඇරගෙන ඒ class එකට අපි මොකක් හරි නමක් දෙමු. හරි යාලුවනේ, මම ඒ class එකට නමක් දෙනවා person කියලා.හරි අපි එහෙනත් person class එක හදමුකෝ මෙන්න මේ විදියට .

```

class Person{

    String name;

    int age;

    String Gender;

    void eat(){

        // statements

    }

    void walk(){

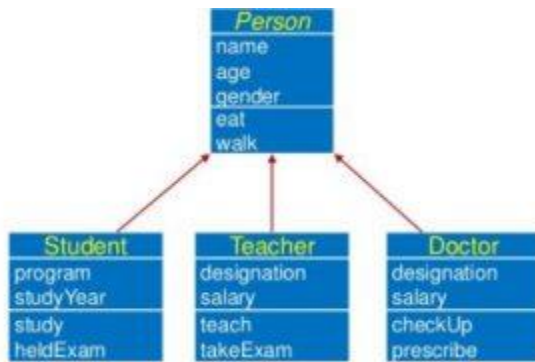
        // statements

    }

}

```

හරි ඔයාලට ජ්‍යෙෂ්ඨව දැන් අපි හදා ගත්තු person class එකේ නියෝජන class තුනටම පොදු attributes හා methods ටික. හරි ඔය හදාගත්තු person class එකට ඔහොමම නියෝජන දීලා බලන්නකෝ එක එක class එකේ ඉතිරි වුන attributes හා method දිහා. හොදට බලුවොත් ඔයාලට ජ්‍යෙෂ්ඨව ඇති යාලුවන් teacher, doctor හා student කියන class තුනේ ඉතිරි වෙලා නියෝජන එකිනෙකට අනන්‍යය වු attributes හා methods කියලා. නවත් විදියකින් කියනවනම් ඒ ඒ class වලට අදාල special attributes හා methods ඉතිරි වෙලා නියෝජන.



හරි අපි දැන් ඒ ඉතිරි වූන attributes හා methods ඒ ඒ අදාල class එකට දාලා අලුත් teacher, student හා doctor කියන class තුන හදා ගමු.

```
class Teacher{

    String Designation;

    int salary;

    void teach(){

        // statements

    }

    void takeExams(){

        // statements

    }

}
```

```
class Doctor{  
  
    String Designation;  
  
    String salary;  
  
    void ChackUp(){  
  
        // statements  
  
    }  
  
    void prescribe(){  
  
        // statements  
  
    }  
}
```

```
class Student{  
  
    String programe;  
  
    int studyYear;  
  
    void study(){
```

```

        // statements

    }

    void heldYear(){

        // statements

    }

}

```

හරි යාලුවනේ දැන් අපි කරන්න ඕනි අමාරුම වැඩ ටික කරලා ඉවරයි.ඒ කියන්නේ , class තුනේ තිබුන පොදු attributes හා methods හදුනා ගෙන ඒ ටික එක් class එකකට දා ගන්නා . ඊට පස්සෙ teacher, student හා doctor කියන class තුනට අදාල special attributes හා methods ටික ඒ ඒ class වල වෙන වෙනම තියා ගන්නා. නව පොඩි දෙයයි ඉතිරි වෙලා තියෙන්නේ.ඒ තමයි අපි අර person කියලා class එකක් හදලා, class තුනටම අදාලව තියෙන පොදු attributes හා methods හැම එකක්ම teacher,doctor හා student කියන class තුනටම වෙන වෙනම ගන්න ඕනි නේද? අන්න ඒකට අපි පාවිච්චි කරනවා extends කියන keyword එක.

```

class Teacher extends Person{

    String Designation;

    int salary;
}

```

```
void teach(){

    // statements

}

void takeExams(){

    // statements

}

}

class Doctor extends Person{

    String Designation;

    String salary;

    void ChackUp(){

        // statements

    }

    void prescribe(){

        // statements

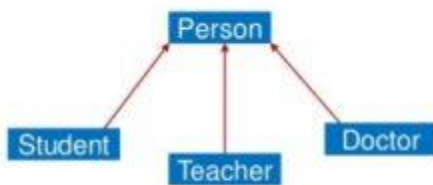
    }

}
```

```
    }  
  
}  
  
class Student extends Person{  
  
    String programe;  
  
    int studyYear;  
  
    void study(){  
  
        // statements  
  
    }  
  
    void heldYear(){  
  
        // statements  
  
    }  
  
}
```

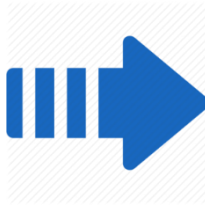
ඔබ්බාම නමයි extends keyword එක පාවිච්චි කරලා එක් class එකක නියෙන attributes සහ methods තවත් class එකකට ගන්නේ.පොදු attributes හා methods නියාගන්න class එකට අපි කියනවා super class එක කියලා.අපෙ උදාහරණයේ හැටියටනම් super class එක වෙන්නෙ person class එකයි.inherit කරපු class එක එහෙමත් නැත්නම් special attributes හා methods නියාගන්න class එකට අපි කියනවා sub class එක කියලා.අපෙ උදාහරණයේ නම් sub classes වෙන්නෙ teacher,doctor හා student කියන class තුනයි.

inheritance වරදී inherit වෙලා තියනවා කියලා අපි පෙන්වන්නෙ sub class එකේ ඉඳලා super class එකට ඊතලයක් ඇඳලා.හරියට මෙන්න මේ වගේ.



ඔයාලා මතක තියා ගන්න ඕනි වැදගත්ම කාරණය නමයි sub class එකක් super class එකකින් inherit වෙද්දී super class එකේ තියෙන privet නොවන සියලුම attributes හා methods sub class එකට එනවා කියන එක.ඊට පස්සෙ ඒ ආව attributes හා methods ඔක්කොම sub class එකේ තබන ඒවා විදියටම තමයි වැඩ කරන්නෙ.කිසිම වෙනසක් නැහැ.ඔබ්බාම ඔබ්බාම නමයි යාලුවනේ inheritance කියලා කියන්නේ.හරි යාලුවනේ, ඔයාලට දැනටමත් තේරෙනවා ඇතිනෙ මේ ක්රමයේ තියෙන වාසිය.කලින් කුමයේදී නම් ඔයාලා class තුනටම පොදු attributes හා methods ටික තුන් පාරක්ම define කලා නමුත් මේ කුමයේදී නම් define කලේ එක පාරයි.ඔබ්බාම ඔබ්බාම නමයි inheritance කියන feature එකේ තියන වාසිය.ඔයාගෙ programe එකේ ස්භාවය අනුව ඔයාලට මේ feature එකේ වාසිය ගන්න පුලුවන්.ඔයාගෙ programe එකේ එක එක class වල පොදු attributes හා methods ගොඩක් තියනවානම් inheritance පාවිච්චි කරන එක ඔයාට පට්ට වාසියි තේද.

හරි අපි තවත් විදියකින් inheritance ගැන හිතමුකො.ඒකට මං මෙන්න මේ වගේ උදාහරණයක් ගන්නමිකෝ.හොඳට මේ යටින් දීලා තියෙන රූපය බලන්නකෝ.



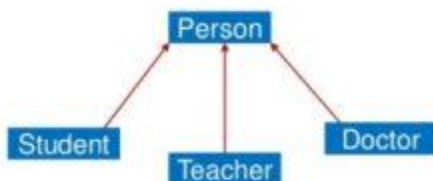
ඇත්තටම මෙකේ නියෙන්නෙ පරම්පරාවල් දෙකක එහෙමත් නැත්නම් generation දෙකක තාක්ෂණයන් එකතු කරලා හදපු ජංගම දුරකථන කීපයක්නෙ. හොදට හිතලා බලන්න කලින් generation එකේ තිබුන දුරකථනයේ features කීපයක් ගැන. ඒ features වලින් දෙකක් අපි ගනිමු call හා message කියලා. අලුත් generation එකේ දුරකථන ගැන බලුවොත් මේ features දෙක එහෙමම නියනවා කියලා ඔයාලට තේරෙනවා. හැබැයි ඊට අමතරව camera, video ,audio කියලා අලුත් features කීපයකුත් ඇවිත් නියනවා. ඒ කියන්නෙ කලින් generation එකේ තිබුන features ඇරගෙන මීලග generation එක දියුණු කරලා හදලා නියනවා. ඔන්න ඔය දේ programming පැත්තෙන් හිතුවොත් මෙන්න මේ වගේ. එක class එකකින් (super class) inherit කරලා තව class එකක් (sub class) හදනවා ඊට වඩා දියුණු තත්වයේ පවතින. inheritance කියන feature එකින් ඔන්න ඔය අදහසත් කියවෙනවා.

ඔන්න ඔය අදහස් දෙක තමයි inheritance එකෙන් කියවෙන්නෙ. ඇත්තටම කියනවනං inheritance කියන්නේ කලින් class එකේ (super class) තිබුන දේවලුත් දෙවලුත් ඇරගෙන ඊට වඩා හොද class එකක් (sub class) එකක් හදන එකටයි . හැමෝටම තේරෙන්න ඇති කියලා හිතනවා අද පාඩම . inheritance වල නියෙන විශේෂ අවස්තා කීපයක් සමගින් මීලග පාඩමේදී හමුවෙමු. ස්තූතියි!

Method Overriding & super keyword

කොහොමද යාලුවනේ! ඔන්න මම අදත් ආවා object oriented concept වල නියෙන අලුත්ම කොටසක් ඇරගෙන. අලුත් පාඩමක් කිව්වට අලුත්ම එකක් නෙවෙයි. ඔයාලා අහලා නියෙන ටිකක් මතක තියෙන පාඩමක් .ඒ තමයි method overriding ගැන. method overriding ගැන කලින් පාඩමක කතා කළා මත්කයි නේද? එතනදි මම කිව්වා method overriding වැඩිපුරම පාවිච්චි වෙන්නේ inheritance වලදී කියලා .inheritance පාඩමේදි method overriding ගැන අපි කතා කතාකරන්න ඉතුරු කළා මත්ක ඇති නේද.. අන්න ඒ ඉතුරු කරපු කොටස ගැන කතා කරන්න තමයි මම මේ පාඩම වෙන් කරන්නේ.

හරි අපි method overriding ගැන කතා කරන්න ඉස්සරින් මම පොඩ්ඩක් කලින් පාඩම පොඩ්ඩක් මතක් කරන්නම්. කලින් පාඩමේදි අපි කතා කලේ object oriented concept එකේ තියන inheritance කියන feature එක ගැනයි. අපේ කලින් පාඩම බැලුවේ නැතිනම් අද පාඩම තේරුම් ගන්න නම් ටිකක් අමාරු වෙයි. ඒ නිසා කලින් පාඩම ඉක්මනින් ගිහින් බලලා එන්නකෝ. කලින් පාඩමේදි අපේ අපි කතා කළා inheritance කියන්නේ මොකක්ද කියලා. ඒ වගේම inheritance භාවිතා වුණු උදාහරණ දෙකකුත් කතා කළා . ඇත්තටම යාලුවනේ මොකද්ද අපි inheritance කියල කිව්වේ. inheritance කියලා අපි කිව්වේ එක class එකක තියෙන method සහ attributes තවත් ක්ලාස් එකකට inherit කරලා කලින් class එකට වඩා දියුණු class එකක් සෑදීමයි. inherit කරපු class එකට අපි කිව්වා super class එක කියලා. inheritance කරල හදපු අලුත් ක්ලාස් එකට අපි කිව්වා කිව්වා sub class එක කියලා. class දෙකක් inherit වෙලා තියනවා කියලා පෙන්වීමට sub class එකේ ඉඳලා super class එකට ඊතලයක් ඇඳලා පෙන්නවා මෙන්න මේ විදිහට.



හරි යලුවනේ මතක් උන නේද inheritance කියන්නේ මොකක්ද කියලා . එහෙනම් අපි බලමු අද පාඩම ගැන.කලින් කිව්ව විදියට අද පාඩම වෙන්නෙ method overriding.ඔන්න යාලුවනේ මේ පාඩම method overloading පාඩමත් එක්ක පටලවා ගන්නනම් එපා.method overloading පාඩමේදී කිව්ව විදියට method overriding කියලා අපි කිව්වේ එක class එකක නියන method එකක් අපිට ඕනි විදියට එකේ body එක වෙනස් කරලා ජරයෝජනයට ගන්න එකටයි.method overriding වැඩිපුරම භාවිතා වෙන්නෙ inheritance වලදී කියලත් අපි ඒ පාඩමේදී කිව්වා මතක ඇති නේද හැමෝටම.එහෙනත් අපි බලමු කොහොමද method overriding , inheritance වලදී භාවිතා වෙන්නේ කියලා.

හරි යාලුවනේ ඔයාලා දන්නවනෙ දැන් super class එකකින් sub class එකක් inherit කරන්නෙ අපි extends කියන keyword එක පාවිච්චි කරල කියලා.මේ විදියට inherit කරද්දි super class එකේ නියන privet නොවන attributes සහ methods සියල්ලක්ම sub class එකට එනවා කියලත් ඔයාලා දන්නවා.ඔන්න ඔය කරුණු ටිකත් ඔලුවේ නියාගෙන පෙඩ්ඩක් මම මේ කියන උදාහරණය බලන්නකෝ.

හරි මම හදනවා class එකක් phone කියලා.ඒකේ attributes විදියට මම ගන්නවා color එක හා price එක.method එකක් විදියට මම ගන්නවා call කියන එක. නවත් class එකක් මම හදනවා SmartPhone කියලා.ඒකේ attributes විදියට මම ගන්නවා color එක, price එක හා storage එක .methods විදියට මම ගන්නවා call හා takePhoto කියලා methods දෙකකුත්.දැන් ඔයාලට හොදට බැලුවොත් ජේනවා මේ class දෙකටම පොදු attributes හා methods නියනවා කියලා..දැන් ඔයාලට පොඩ් අදහසක් එනවා නේද එක class එකක් අනික් class එකෙන් inherit කලොත් හොදයි කියලා.හරි එහෙනම් අපි inherit කරමු.

```
class Phone{  
  
    String color;  
  
    int price;  
  
  
    void call(){
```

```

        System.out.println("This phone can take audio calls only ! ");

    }

}

class SmartPhone extends Phone{

    String storage;

    void takePhoto(){

        System.out.println("This phone can take quality photos ! ");

    }

}

```

හරි ඔබ්න ඔහොම තේද යාලුවනේ SmartPhone කියන class එක phone කියන class එකින් inherit කලාම එන්නේ. මොකක්ද එතකොට අපේ super class එකයි sub class එකයි වෙන්නේ? අපේ super class එක වෙන්නේ Phone class එක. sub class එක වෙන්නේ SmartPhone කියන class එකයි. inherit කිරීමක් කරපු නිසා super class එකේ තිබුණ attributes සහ methods ඔක්කොම sub class එකට ආවා තේද.. හරි යාලුවනේ දැන් මං කියන දේ හොඳට සිහිමිව තේරුම් ගන්නකෝ. හරි ඔයා හිතන්න දැන් Phone කියන class එකෙන් හදන object එකකට පුලුවන් call කියන method එකට කතා කලාම audio call විතරයි ගන්න පුලුවන් කියලා. නමුත් SmartPhone කියන class එකෙන් හදන object එකකට තියන call කියන method එකට කතා කලාම audio and video call කියන දෙකම ගන්න පුලුවන් කියලා හිතන්න. (SmartPhone කියන class එක ඇතුලේ අපිට ජේන්න call කියලා method එකක් නැති වුනාට inheritance කරලා තියන නිසා super class එකේ තියන call කියන method එක SmartPhone කියන class

එකට ඇවිත් තියෙන්නේ.)දැන් අපි Phone කියන class එකෙයි SmartPhone කියන class එකෙයි object දෙකක් හදලා call කියන method එකට කතා කරමු.

```
class Phone{

    String color;

    int price;

    void call(){

        System.out.println("This phone can take audio calls only ! ");

    }

}

class SmartPhone extends Phone{

    String storage;

    void takePhoto(){

        System.out.println("This phone can take quality photos ! ");

    }

}
```

```

class CreatingObjects{

    public static void main(String a[]){

        Phone phone1=new Phone();

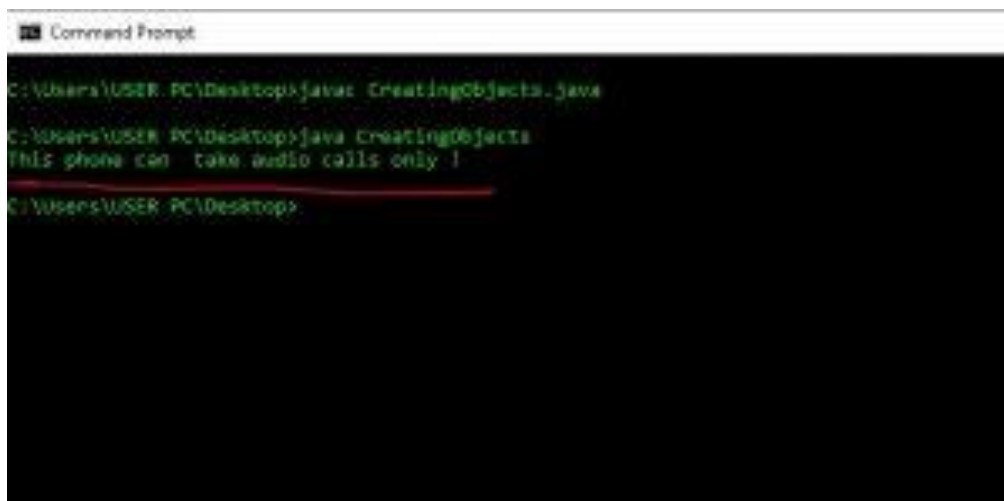
        phone1.call();

    }

}

```

output එක දිහාත් හොදට බලන්නකෝ.මේ නියෝගෙන Phone කියන class එකින් හදපු object එකේ call method එකට කතා කලාම ලැබෙන output එක.



```

C:\Users\USER\PC\Desktop>javac CreatingObjects.java
C:\Users\USER\PC\Desktop>java CreatingObjects
This phone can take audio calls only !
C:\Users\USER\PC\Desktop>

```

මේ නියෝගෙන SmartPhone කියන class එකින් හදපු object එකේ call method එකට කතා කලාම ලැබෙන output එක.

```

class CreatingObjects{

    public static void main(String a[]){

```

```

        SmartPhone smartphone1=new SmartPhone();

        smartphone1.call();

    }

}

```

```

C:\Users\USER_PC\Desktop>java CreatingObjects.java
C:\Users\USER_PC\Desktop>java CreatingObjects
This phone can take audio calls only !
C:\Users\USER_PC\Desktop>

```

Phone class එකේ call method එකට කතා කලාම ලැබෙන output එකනම් හරි නේද..SmartPhone class එකේ call method එකට කතා කලාම ලැබෙන output එක නම් වැරදියි නේද..SmartPhone class එකේ call method එකට කතා කලාම මොකක් හරි output එකක් ලැබුනනෙ ..ඒ කියන්නෙ අපි inheritance කරපු එකේ ප්රශ්නයක් නෑ.call method එක SmartPhone Class එක ඇතුලට ඇවිත් තියෙන්නේ.නමුත් ඇවිත් තියෙන්නෙ Phone කියන class එකේ තිබුන call method එකේ body එකත් ඇරගෙන නේද..ඒ නිසා තමයි අපිට SmartPhone class එකේ තියන call method එකට කතා කලාම Phone class එකේ තිබුන call method එකේ body එක execute වෙලා පෙන්නුවේ .එහෙනම් අපිට SmartPhone class එකට අවශ්‍ය විදියට call method එක execute වෙන්න ඕනි නම් ,SmartPhone class එකේ call method එකේ body එක මෙන්න මේ විදියට වෙනස් කලානම් හරි නේද..

```

class Phone{

```

```
String color;

int price;

void call(){

    System.out.println("This phone can take audio calls only ! ");

}

}

class SmartPhone extends Phone{

    String storage;

    void takePhoto(){

        System.out.println("This phone can take quality photos ! ");

    }

    void call(){

        System.out.println("This phone can take audio calls and video calls ! ");

    }

}
```

```

}

class CreatingObjects{

    public static void main(String a[]){

        SmartPhone smartphone1=new SmartPhone();

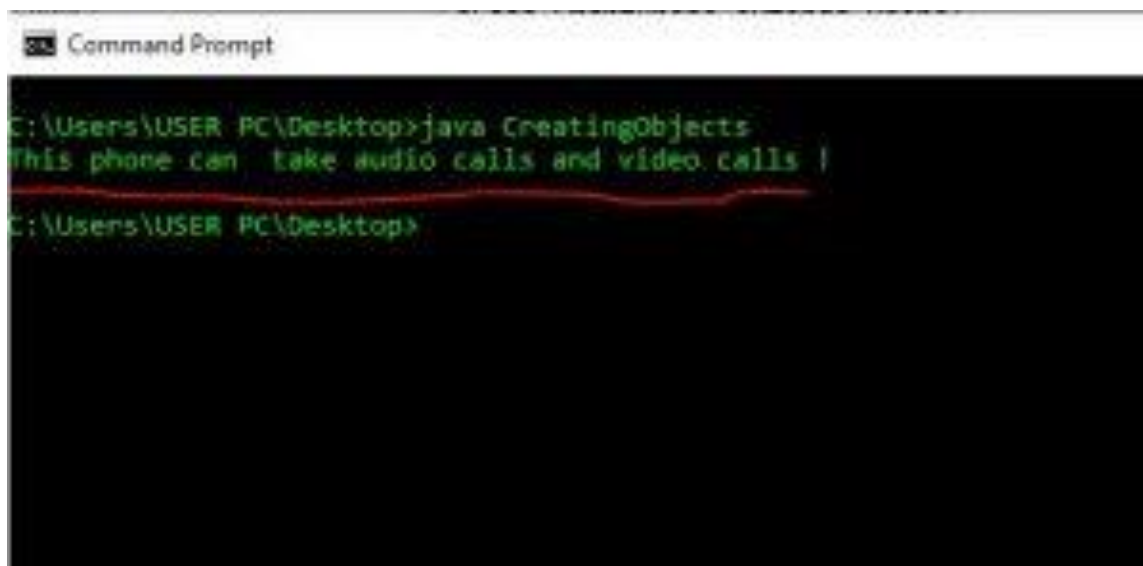
        smartphone1.call();

    }

}

```

දැන් බලන්නකෝ output එක දිහා. දැන්නම් හරි නේද..



```

Command Prompt

C:\Users\USER PC\Desktop>java CreatingObjects
This phone can take audio calls and video calls !
C:\Users\USER PC\Desktop>

```

ඔන්න ඔකට තමයි යාලුවනේ method overriding කියලා කියන්නෙ.කෙටියෙන්ම කිව්වොත් එක class එකකින් තවත් class එකක්

inherit කලාම super class එකේ නියන methods , sub class එකට අවශය ආකාරයට ඒවායේ body එක වෙනස් කරලා පාවිච්චි කරන සිද්ධියට තමයි method overriding කියලා කියන්නේ.ඔන්න ඔයාලට මං ආයෙන් කියනවා method overloading එක්ක මේක පටලව ගන්නනම් එපා.**එකම නමින් methods ගොඩාක්**ඒවායේ parameter signature එක වෙනස් කරලා එකම class එකක් ඇතුලේ define කරන එකට තමයි අපි කිව්වේ method overloading කියලා.**එකම method එකක** body එක තමන්ට ඕනි විදියට වෙනස් කරලා පාවිච්චි කරන එකට තමයි අපි method overriding කියලා කිව්වේ.ඔන්න ඕකම තමයි මේ දෙක අතර නියන වෙනස.හරි තේරුනා නේද යාලුවනේ method overriding කියන්නේ මොකක්ද කියලා.

හරි එහෙනත් අපි බලමු method overriding වලදි හමුවන keyword එකක් ගැන.ඒ තමයි super keyword එක.ඇත්තටම යාලුවනේ super keyword එකින් කරන්නේ යට නියන දෙයක් උඩට ඇදලා දෙන එක.☺☺☺ තේරුන් නෑ නේද කිව්ව දේ..හරි ඔයාලා දන්නවනේ inherit වෙලා නියන class එකක super class එකේ method එකක body එක වෙනස් කරලා sub class එකේදි ඒ method එක පාවිච්චි කරන විදිය ගැන (method overriding).දැන් ඔයා හිතන්න ඒ විදියට sub class එකේ නියන method එකක body එක වෙනස් කරගෙන යද්දී ඔයාට ඕනි වෙනවා super class එකේ නියන ඊට අදාළ method එකේ body එක..එහෙම කරන්නට මොකද අර sub class එකේ override කරනු method එක ඇතුලෙ ඉදලා super class එකේ ඒ නමම නියන method එක call කරන්න ඔනි නේද..තේරුනා මදි වගේනම් කලින් උදාරණයම ගමුකෝ.

මට ඕනි දැන් sub class එකේ call method එකේ නියන පලමු statement එක output එක විදියට දීලා ඊලග statement එකේදී super class එකේ call method එකට කතා කරන්න.ඒ කියන්නේ මෙන්න මේ විදියට.

```
class Phone{  
  
    String color;  
  
    int price;  
  
  
    void call(){
```

```
        System.out.println("This phone can take audio calls only ! ");
    }
}

class SmartPhone extends Phone{

    String storage;

    void takePhoto(){

        System.out.println("This phone can take quality photos ! ");

    }

    void call(){

        System.out.println("This phone can take audio calls and video calls ! ");

        call();

    }

}
```

```

class CreatingObjects{

    public static void main(String a[]){

        SmartPhone smartphone1=new SmartPhone();

        smartphone1.call();

    }

}

```

output එකත් බලන්නකෝ.

```

Command Prompt
Exception in thread "main" java.lang.StackOverflowError
at sun.nio.cs.SingleByte.withResult(Unknown Source)
at sun.nio.cs.SingleByte.access$000(Unknown Source)
at sun.nio.cs.SingleByte$Encoder.encodeArrayLoop(Unknown Source)
at sun.nio.cs.SingleByte$Encoder.encodeLoop(Unknown Source)
at java.nio.charset.CharsetEncoder.encode(Unknown Source)
at sun.nio.cs.StreamEncoder.implWrite(Unknown Source)
at sun.nio.cs.StreamEncoder.write(Unknown Source)
at java.io.OutputStreamWriter.write(Unknown Source)
at java.io.BufferedWriter.flushBuffer(Unknown Source)
at java.io.PrintStream.write(Unknown Source)
at java.io.PrintStream.print(Unknown Source)
at java.io.PrintStream.println(Unknown Source)
at SmartPhone.call(CreatingObjects.java:20)
at SmartPhone.call(CreatingObjects.java:21)
at SmartPhone.call(CreatingObjects.java:21)
at SmartPhone.call(CreatingObjects.java:21)

```

error එකක් දෙනවා නේද interpreter එකෙන්.ඔන්න ඔය එන error එක වලක්වා ගන්න තමයි අපි super keyword එක භාවිතා කරන්නේ.

```

class Phone{

    String color;

    int price;

```

```
void call(){

    System.out.println("This phone can take audio calls only ! ");

}

}

class SmartPhone extends Phone{

    String storage;

    void takePhoto(){

        System.out.println("This phone can take quality photos ! ");

    }

    void call(){

        System.out.println("This phone can take audio calls and video calls ! ");

        super.call();

    }

}
```

```

}

class CreatingObjects{

    public static void main(String a[]){

        SmartPhone smartphone1=new SmartPhone();

        smartphone1.call();

    }

}

```

output:



```

C:\Users\USER PC\Desktop>javac CreatingObjects.java
C:\Users\USER PC\Desktop>java CreatingObjects
This phone can take audio calls and video calls !
This phone can take audio calls only !
C:\Users\USER PC\Desktop>

```

දැන් හරි නේද යාලුවනේ. ඔන්න ඔක තමයි super keyword එකේ කාර්යය. ඇත්තටම කියනවනම් ඔය super keyword එක කරන්නේ මම අර කලින් කිව්වා වගේ යට තියන දෙයක් උඩට ඇදීමක්. super class එකේ

නියත method එකක් sub class එකේදී override කලාම වෙනතෙ sub class එකේ override කරපු method එකෙන් super class එකේ නිවුන ඒ method එක වැහිලා යන එකයි.super keyword එකින් කරන්නෙ ඒ වැහිලා ගියපු method එක අයෙත් උඩට ඇදලා පෙනෙන එකයි.අන්න ඒකයි මං කිව්වෙ super keyword එකෙන් කරන්නෙ යට නියත දෙයක් උඩට අදින එක කියලා.

හරි යාලුවනේ තේරුනා නේද method overriding ගැනයි super keyword එකේ භාවිතය ගැනයි.මොකක් හරි අවුල්ක් ආවොත් කොමෙන්ටුවක් දාන්න ඕනි හොඳේ ☺☺☺.අදට පාඩම අවසන්.object oriented concept එකේ නියත තවත් feature එකක් සමගින් මිලග පාඩමෙන් හමුවෙමු.සුභ දවසක්.ස්තූතියි!

Upcasting & Downcasting

ආයුබෝවන්! ඔන්න යාලුවනේ මං අදත් ආවා object oriented concept එකේ නියත අලුත් පාඩමක් එක්ක.සුපුරුදු විදියට ඔයාලා හැමෝම සාදරයෙන් පිළිගන්නවා.අද පාඩම පටන් ගන්න ඉස්සරින් කලින් පාඩම පොඩ්ඩක් මතක් කරලා ඉන්නමිකෝ.කලින් පාඩමේදී අපි කතා කලා method overriding ගැනයි super keyword එක ගැනයි.method overriding කියලා අපි කිව්වේ මොකක්ද...method overriding කියලා අපි කිව්වේ super class එකේ නියත method එකක් sub class එකේදී ඒ method එකේ body එක වෙනස් කරලා පාවිච්චි කරන එකට.super keyword එකෙන් කරන්නේ , super class එකේ method එකක් sub class එකේදී override කලාම ඒ override කරපු එකෙන් super class එකේ ඒ method එක සැගවෙනවා. ඒ සැගවුනු super class method එක එලියට ඇදලා ගන්න නමයි අපි super keyword එක පාවිච්චි කලේ.ඔන්න ඕවා දෙක ගැන නමයි අපි කලින් පාඩමේදී කතා කලේ..තවත් හරියට මතක නැතිනම් අපේ කලින් පාඩම ගිහින් බලන් එන්නකෝ.

හරි එහෙනම් ඕං මං අද පාඩම පටන් ගන්නවා.oop වල නියෙන මිලග feature එක හරි හැටි තේරුම් ගන්න නම් මේ පාඩම හරියටම දැනගෙන ඉන්න වෙනවා..අද පාඩම හරි හැටි තේරුම් ගන්න නම් inheritance පාඩම හොදටම මතක තියෙන්න ඕනි.. inheritance කියන්නෙ මොකද්ද කියලා පොඩ්ඩ හරි අමතක වීමක් තියනවානම් ඒ පාඩමත් ඉක්මනින් බලන් එන්නකෝ..හරි ඉස්සරෝම කියන්නම් මං upcasting කියන්නෙ මොකක්ද කියලා.upcasting කියලා කියන්නෙ, class දෙකක් inherit කලාම super class

එකකුයි sub class එකකුයි හැදෙනවනේ ඒ හැදෙන sub class object එකක් super class reference variable එකක store කර ගන්න එකට නමයි upcasting කියලා කියන්නේ.

හරි මං ගන්නමිකො මෙන්න මේ උදාහරණය ඔයාලට මේක පැහැදිලි කරන්න.මම හදනවා class දෙකක්..එකක් doctor කියලා ගන්නවා අනික teacher කියලා ගන්නවා.මෙන්න මේ classes දෙකේ නියත attributes හා methods මේ විදියට define කරලා teacher class එකයි doctor class එකයි හදා ගන්නවා.



දැන් ඔයාලට පෙනවා මේ class දෙකේම පොදු attributes හා methods නියනවා කියලා.ඒ ටික පොදු class එකකට මං දා ගන්නවා.ඒ දා ගන්නා පොදු class එකට මං නමක් දෙනවා person කියලා..ඒ person class එකෙන් doctor හා teacher කියන classes දෙකම inherit කරනවා.එතකොට code එක මෙන්න මේ විදියයි තේද..

```
class Person{

    String name;

    int age;

    String Gender;
```

```
void eat(){

    // statements

}

void walk(){

    // statements

}

}

class Teacher extends Person{

    Teacher(){

        System.out.println("teacher object is created !");

    }

    String Designation;

    int salary;
```



```
void teach(){

    // statements

}

void takeExams(){

    // statements

}

}

class Doctor extends Person{

    String Designation;

    String salary;

    void ChackUp(){

        // statements

    }

    void prescribe(){

        // statements

    }

}
```

```

    }

}

```

ඔබ්බට එනකන් තේරුනානෙ තේද...ඔබ්බට එනකන්ම තිබ්බෙ inheritance ගැන..අන්න ඒකයි මං කලින්ම කිව්වෙ මේ පාඩම තේරුම් ගන්න නම් inheritance පාඩම හොදට මතක තියෙන්න ඔබ්බ කියලා..හරි අපි දැන් class දෙක inherit කලානෙ.එනකොට අපේ super class එක වෙනවා person class එක.sub classes වෙනවා teacher class එකයි doctor class එකයි.එනතට එනකන් නම් අවිලක් නැනෙ තේද..එහෙනම් අපි ඊලග පියවරට යං.අපිට දැන් sub classes දෙකක් තියනවනේ.ඒකෙන් එකක් මං ගන්නවා...ඔයාලට මේ දෙකෙන් කැමති එකක් ගන්න පුලුවන්..හරි මං ගන්නවා teacher කියන class එක.නව super class එකත් මං ගන්නවා. එනකොට මං ගන්නා super class එකයි teacher කියන sub class එකයි.දැන් මං හදා ගන්නවා අපේ super class එකේ reference එකක් මෙන්න මේ විදියට.

```

class Upcasting&Downcasting{

    public static void main(String a[]){

        Person person1;

    }

}

```

හරි ඔන්න ඔය විදියට super class reference variable එකක් හදා ගන්නා. ඊට පස්සෙ මං අපි අර ගන්න sub class එක තියෙන්නෙ, ඒ කිව්වෙ අපෙ අර teacher class එක, අන්න ඒ teacher class එකේ object එකක් හදලා අපි අර හදා ගන්න super class reference variable එකට දා ගන්නවා මෙන්න මේ වගේ.

```
class Upcasting&Downcasting{

    public static void main(String a[]){

        Person person1 = new Teacher();

    }

}
```

ඔන්න ඔකට තමයි upcasting කියන්නේ. කෙටියෙන් කියනවානම්, upcasting කියලා කියන්නෙ, class දෙකක් inherit කලාම super class එකකුයි sub class එකකුයි හැදෙනවනේ ඒ හැදෙන sub class object එකක් super class reference variable එකක store කර ගන්න එකට තමයි upcasting කියලා කියන්නේ.

ඇත්තම නව පොඩ්ඩක් practicaly හිතලා බලන්නකෝ මේක ගැන. අපි cast කරන්න ගත්තේ super class එක විදියට Person class එකයි Sub class එක විදියට teacher class එකයි. upcasting වලදි කල්ලේ sub class එකේ object

එකක් හදාගෙන ඒ object එක super class reference variable එකකට දා ගන්න එකතේ.ඔන් ඕක පොඩ්ඩක් මෙහෙමත් හිතන්නකෝ... අපෙ sub class එකේ object එකක් හදනවා කියන්නේ teacher object එකක් හදන එකටනේ..අපෙ උදාහරනයේ හැටියට super class reference එකක් කියන්නේ person කෙනෙක් නියා ගන්න පුලුවන් variable එකක් හදනවා කියන එකටයි.හැබැයි teacher කියන්නෙන් තවත් එක්තරා ආකාරයක person කෙනෙක් නේද...තවත් විදියකින් කියනවානම් teacher කියන්නේ person කියන ලොකු කුලකයේ නියන පොඩි උපකුලකයක්...ඒ කියන්නේ person කෙනෙක් නියා ගන්න හදපු reference variable එකකට teacher කෙනෙක් නියා ගන්න පුලුවන්..ඇයිසේ කියනවා නම් teacher කියන්නෙන් එක්තරා විදියක person කෙනෙක් නිසා..ඔන්න ඔය super class reference variable එකක sub class object එකක් නියා ගන්න එකට අපි කියනවා upcasting කියලා .

පොඩ්ඩක් හිතන්න ඕකෙ අනික් පැත්ත ගැන..ඒ කියන්නේ..sub class reference variable එකක super class object එකක් store කරන්න පුලුවන්ද කියලා.හරියට මෙන්න මේ වගේ.

```
Teacher teacher1 = new Person();
```

අපෙ උදාහරනයේ හැටියට නම් teacher කෙනෙක් නියා ගන්න reference variable එකක් හදලා ඒකේ person කෙනෙක් store කරනවා වගේ දෙයක්.ඇත්තටම පුලුවන්ද මේ වැඩේ කරන්න???.බැහැ නේද...ඇයි යාලුවනේ හිතලා බලන්නකෝ teacher කෙනෙක් නියා ගන්න ඔයා හදනවා reference variable එකක්. ඒකෙ ඕනෑම person කෙනෙක් දාන්න පුලුවන්ද? බැහැ නේද..කොටින්ම කිව්වොත් teacher කියන කුලකය ඇතුලේ හැම person කෙනෙක්ම ඉන්නවද?? නැහැ නේද...

```
Teacher teacher1 = new Person();
```



ඔන්න ඔය ඡේතුව නිසා තමයි බැරි sub class reference variable එකක super class object එකක් නියා ගන්න..එහෙනම් යාලුවනේ මොකද්ද downcasting කියන්නේ????

අන්තර්ගතය යාලුවනේ downcasting කියන්නේ sub class reference variable එකක super class object එකක් store කරන එකට නම් නෙවෙයි..downcasting කියන්නේ, upcasting කරලාම sub class object එකක් super class reference variable එකක store කරනවනේ ඒ store කරපු object එක අයෙම sub class reference එකකට දාන එකට නමයි downcasting කියන්නේ.ඒ කියන්නේ අපේ උදාහරණයේ හැටියට නම් , teacher class එකෙන් හදාගන්න object එක store කරේ person raference variable එකකනේ...ඒ විදියට store කරගන්න sub class object එක අයෙත් sub class reference variable එකකම store කර ගන්න එකට.code එක ලිව්වොත් මෙන්න මේ වගේ..

```
class Upcasting&Downcasting{

    public static void main(String a[]){

        Person person1 = new Teacher() ; // upcasting
    }
}
```

```

        // downcasting

        Teacher teacher1 = person1 ;

    }

}

```

output :



```

C:\Users\USER\PC\Desktop>javac test.java
test.java:49: error: incompatible types: Person cannot be converted to Teacher
        Teacher teacher1 = person1 ;
                        ~~~~~
1 error
C:\Users\USER\PC\Desktop>

```

output එක බලන්නකෝ..error එකක් දෙනවා නේද.අත්තටම මෙතනදී වෙනේ copiler එක හිතනවා super class reference variable එකේ store කරලා තියෙන්නේ super class object එකක් කියලා.sub class reference

variable එකක super class variable එකක් store කරන්න බැරි නිසා නමයි එහෙම error එකක් දුන්නේ.

ඔන්න ඔය error එක නැති කර ගන්න පුලුවන් casting කරලා මෙන්න මේ විදියට..

```
class Upcasting&Downcasting{

    public static void main(String a[]){

        Person person1 = new Teacher() ; // upcasting

        // downcasting

        Teacher teacher1 = (Teacher) person1 ;

    }

}
```

output:

```
Command Prompt
C:\Users\USER PC\Desktop>javac test.java
C:\Users\USER PC\Desktop>java test
Teacher object is created !
C:\Users\USER PC\Desktop>
```

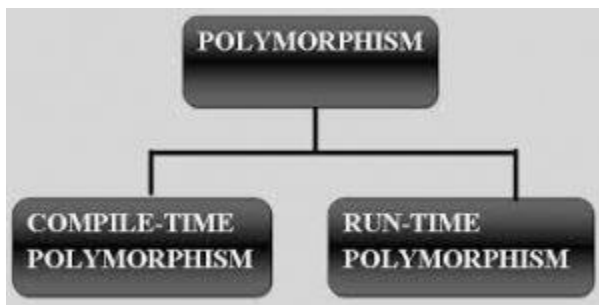
දැන්නම් හරි තේද යාලුවනේ. ඇත්තටම casting කරන එකෙන් මෙතනදි වෙනතෙ super class reference variable එකේ store කරලා තියෙන්නේ sub class එකෙන් හදපු object එකක් කියලා compiler එකට පෙන්නන එක. sub class object එකක් sub class reference variable එකක store කරන එක අවුලක් නෑනෙ.. අන්න එනකොට තමයි error එකක් නැති out put එකක් දෙනවා. ඔන්න ඔකට තමයි downcasting කියලා කිව්වේ.

Polymorphism

කොහොමද ඉතින් ! oop වල මීලඟ පාඩම බලාපොරොත්තුවෙන් සිටි යාලුවෝ හැමෝම සාදරයෙන් පිළිගන්නවා.. පුරුදු විදියටම මං කලින් පාඩම පොඩ්ඩක් මතක් කරලා ඉන්නම්කෝ... කලින් පාඩමේදී අපි කතා කලා upcasting හා downcasting කියන්නේ මොකද්ද කියලා හා කොහොමද අපි upcasting හා downcasting කරන්නේ කියලා. මොකක්ද අපි upcasting කිව්වේ?? upcasting කියලා අපි කිව්වේ... sub class object එකක් හදලා ඒක super class reference variable එකක store කරන එකට.. එහෙම super class reference variable එකක store කරපු sub class object එක නැවත sub class reference variable එකක store කරන එකට අපි කිව්වා downcasting කියලා. ඔන්න ඔය දෙක ගැන තමයි කලින් පාඩමේදී අපි කතා කලේ. මතක නැහැ වගේ නම් කලින් පාඩම පොඩ්ඩක් බලන් එන්න වෙනවා. නැත්නම් අද පාඩම තේරුම් ගන්න ටිකක් අමාරු වෙනවා. 😊

හරි එහෙත්ම ඔන්ත ඔය කලින් පාඩමන් ඔලුවෙ නියාගෙනම අපි බලමුකෝ අද පාඩම මොකක්ද කියලා..අද පාඩම නමයි Polymorphism . oop වල නියෙන ඉතා වැදගත් පාඩමක් වගේම ගොඩ දෙනෙක් හරි හැටි තේරුම් නොගන්නා හරි ලේසි පාඩමක්.මේ blog post එකෙන් අපි හරියටම තේරුම් ගමු Polymorphism කියන්නේ මොකද්ද කියලා.

Polymorphism කියන වචනය හැදිලා නියෙන්නේ “Poly” සහ “morphs”කියන ග්රීක වචන දෙකේ එකතුවෙන්.අන්තටම මේ polymorphism කියන එකේ සරල තේරුම් නමයි එක දෙයක් එම එහෙමත් නැත්නම් action එකක් ක්රම කිහිපයකින් සිදුකිරීමයි. polymorphism ජ්රධාන වශයෙන් compile time polymorphism සහ runtime polymorphism කියලා කොටස් දෙකකට වෙන් කරන්න පුළුවන් .



කලින් පාඩමකින් අපේ සාකච්ඡා කලා නිදේද method overloading ගැන. method overloading අපේ යාලුවන් ඔක්කොටම මතක ඇති කියලා හිතනවා.. method overloading පාවිච්චි කරලා එක action එකක් ක්රම කිහිපයකින් කරනවට අපි කියනවා compile time polymorphism කියලා කියන්නේ. හරිම ලේසියි තේද...

නමුත් අපි අද කතා කරන්න යන්නේ runtime polymorphism ගැන. එහෙමත් නැත්නම් තවත් විදියකින් කියනවා නම් Dynamic method dispatch කියන එක ගැන.ඔන්ත ඔක නමයි අපේ අද මාතෘකාව. හරි එහෙත්ම අපි බලමු මොකද්ද මේ runtime polymorphism(Dynamic Method Dispatch) කියන්නේ කියලා

අන්තටම මේ පාඩම තේරුම් ගන්න නම් , inheritance,method overriding හා upcasting කියන පාඩම් තුනම ඕනි වෙනවා.ඔන්ත ඔය පාඩම් තුනේ පොඩි හරි අවලක් නියනවානම් මේ පාඩමන් ටිකක් අමාරු වෙයි..ඒ නිසා ඔන්ත ඔය පාඩම් තුන හරියටම තේරුම් ඇරන් ඉන්න ඕනි ඉස්සරෝම.

හරි runtime polymorphism(Dynamic Method Dispatch) කියලා කෙටියෙන්ම කියන්නමකෝ මම. runtime polymorphism කියලා කියන්නේ sub class object store කර ගන්න super class reference variable එකක් හදාගෙන (upcasting) ඒ reference variable එක පාවිච්චි කරලා sub class එකේ හෝ sub class වල තියන attributes හා methods වලට call කරන එකට.ඇත්තටම මේ super class reference variable එක ඕනෑම sub class එකක object එකක් store කරන්න ගන්න පුලුවන් පොදු variable එකක්.තේරුනේ නෑ නේද...හරි හරි කලබල වෙන්න එපා මං උදාහරණයක් ඇරගෙන පැහැදිලි කරන්නම්. වාහන වර්ග තුනක් ගන්නවා මං car, truck හා motorcycle කියලා.හරි අපි එහෙනන් ඉස්සරෝම class තුන හදලා ඉමුකෝ.

```
class Car{

    int numberOfWheels;

    String brand;

    String color;

    int speed;

    void park(){

        System.out.println("car is park at second flow !");

    }

    void driveFast(){

        // statement ;

    }
```

```
}
```

```
class Truck{
```

```
    int numOfWeels;
```

```
    String brand;
```

```
    String color;
```

```
    String weight;
```

```
    void park(){
```

```
        System.out.println("Truck is park at ground flow !");
```

```
    }
```

```
    void hoist(){
```

```
        // statement ;
```

```
    }
```

```
}
```

```

class Motorcycle{

    String brand;

    String color;

    int year;

    int numOfWeeks;

    void park(){

        System.out.println("Truck is park at garden !");

    }

    void ride(){

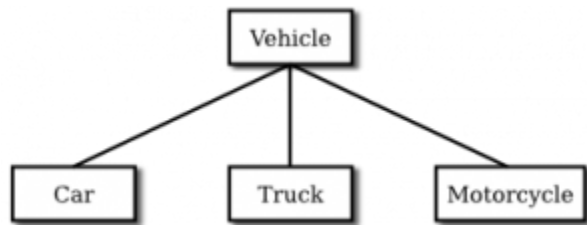
        // statement ;

    }

}

```

ඔබේ ඔබ්බේ නමයි class තුළ එන්නේ...class තුළේම පොදු attributes හා methods තියෙන නිසා පොදු attributes හා methods එක class එකකට ගනු...ඒ ගන්න class එකට මං නමක් දෙනවා vehicle කියලා.



අපේ ඒ vehicle class එකෙන් car,truck හා motorcycle කියන class තුන inherit කරමු...එනකොට අපේ super class එක වෙනවා vehicle කියන class එක. sub classes වෙනවා car,truck හා motorcycle කියන class තුනම..ඔන්න ඔය ටික තේරුනා නේද?...ඔතෙන්ට එනකල්ම නිවුනේ inheritance පාවිච්ඡි. class තුනටම park method එක පොදු නිසා ඒකත් super class එක තුළට දලා sub class වලදී park method එක override කරනවා . code එක මේ වගේ නේද inherit කරලා ඉවර උනාම.

```
class Vehicle{

    int numberOfWheels;

    String brand;

    String color;

    void park(){

        // abstract body ;

    }

}

class Car extends Vehicle{
```

```
int speed;
```

```
void park(){
```

```
    System.out.println("Truck is park at second flow !");
```

```
}
```

```
void driveFast(){
```

```
    // statement ;
```

```
}
```

```
}
```

```
class Truck extends Vehicle{
```

```
    int weight;
```

```
void park(){
```

```
    System.out.println("Truck is park at ground flow !");
```

```
}
```

```
void hoist(){  
    // statement ;  
}  
  
}  
  
class Motorcycle extends Vehicle{  
    int year;  
  
    void park(){  
        System.out.println("Truck is park at garden !");  
    }  
  
    void ride(){  
        // statement ;  
    }  
  
}
```

හරි ඔන්න ඔය විදියට අපි vehicle class කියන super class එකෙන් inherit කරලා car, truck හා motorcycle කියන sub class තුන හදාගෙන, vehicle එකේ තිබුන park method එක එක් එක් sub class එකේදී override කරලත් code එක ලියා ගන්නා..

මං runtime polymorphism කියලා කිව්වේ මොකද්ද?..runtime polymorphism කියලා කිව්වේ sub class objects නියා ගන්න පුලුවන් එක පොදු reference variable එකක් හදාගෙන ඒ ඒ sub class වල නියන methods හා attributes වලට ඒ පොදු reference variable එක හරහා call කිරීමනේ....හරි එහෙනත් අපි පොදු reference variable එකක් හදා ගමු..කොහොමද අපි sub class objects නියා ගන්න පුලුවන් පොදු variable එකක් හදා ගන්නේ.... දැන් ඔයාලට මනක් වෙනවා නේද කලින් කරපු upcasting පාඩමඅන්න ඒ පාඩමේ විදියට super class reference variable එකක් හදා ගන්නොත් පුලුවන් නේද sub class ඕනෑම object නියාගන්න.

```
class MyClass{  
  
    public static void main(String a[]){  
  
        Vehicle vehicle1;  
  
    }  
  
}
```

හරි එහෙනත් පොදු reference variable එකකුත් හදා ගන්නා...දැන් sub class objects ඒ හදා ගත්තු පොදු variable එකේ store කරලා ඒ ඒ object එකේ attributes හා methods වලට call කරමු මෙන්න මේ විදියට..

```
class MyClass{
```



```

public static void main(String a[]){

    Vehicle vehicle1;

    vehicle1=new Car(); // storing car object in super class reference variable..

    vehicle1.park();

    vehicle1=new Truck(); // storing Truck object in super class reference variable..

    vehicle1.park();

    vehicle1=new Motorcycle(); // storing motorcycle object in super class reference variable..

    vehicle1.park();

}

}

```

output එක දිහාත් පොඩ්ඩක් බලන්නකෝ...methods call වෙලා නියනවා නේද...

```
Command Prompt
C:\Users\USER\PC\Desktop>javac MyClass.java
C:\Users\USER\PC\Desktop>java MyClass
Truck is park at second flow !
Truck is park at ground flow !
Truck is park at garden !
C:\Users\USER\PC\Desktop>
```

හරි ඔන් න ඔකට නමයි runtime polymorphism(Dynamic Method Dispatch) කියලා කියන්නේ...කෙටියෙන් කියනවනං , sub classes objects නියා ගන්න පුලුවන් super class reference variable එකක් හදාගෙන ඒ reference variable එක පාවිච්චි කරලා sub class වල නියෙන attributes හා methods වලට call කරන එකටයි.runtime polymorphism(Dynamic Method Dispatch) කියන එකේ අදහස ඔන් න ඔකයි..පාඩම තේරුනා කියලා හිතනවා...

Encapsulation

ආයුරෝවන් යාලුවන් හැමෝටම..object oriented concept එකේ මිලග පාඩමත් සමග එකතු වෙන යලුවන් හැමෝම සාදරයෙන් පිලිගන්නවා.අද පාඩමේදී මම කතා කරන්න බලාපොරොත්තු වෙනවා oop වල නියන අලුත්ම කොටසක් ගැන.ඒ නමයි encasulation.encapsulation ගැන කතා කරන්න ඉස්සරින් සපුරුදු විදියටම කලින් පාඩම පොඩ්ඩක් මතක් කරලා ඉන්නමි.

කලින් පාඩමේදී අපි කතා කලේ polymorphsm ගැන ...මොකද්ද අපි poymorphsm කියලා කිව්වේ...poymorephsm කියලා අපි කිව්වේ, එක action එකක් කුම කිහිපයකින් කරන එකට නේද...polymorphsm ජර්ධන වශයෙන් compile time polymorephsm(static polymorphsm) හා run time polymorphm(dinamic method dispatch) කියලා කොටස් දෙකකට බෙදෙනවා කියලත් කිව්වා ..එක action එකක් method overloading පාවිච්චි කරලා ක්රම කිහිපයකින් කරන එකට අපි කිව්වා compile time poymorephsm (static poymorephsm) කියලා. run time polymorephsm එහෙමත් නැත්නම් dinamic method dispactch කියලා අපි කිව්වේ මොකද්ද.....run time poymorephsm

එහෙමත් නැත්නම් dynamic method dispatch කියලා අපි කීව්වේ sub classes වල objects store කර ගන්න super class reference variable එකක් හදාගෙන ඒ හදාගන්නා reference variable එක පාවිච්චි කරලා sub class වල නියත attributes හා methods වලට call කරන එකට. ඒ super class reference variable එක sub class objects store කරන්න නියත පොදු variable එකක් කියලත් අපි කීව්වා ... හරි දැන් මතක් වුනා නේද polymorphism ගැන.... තව දුරටත් විස්තර ඇතුළු polymorphism ගැන බලන්න ඕනි නම් මෙතනින් අපේ කලින් පාඩමට ගිහින් බලලා එන්න...

හරි එහෙනත් අපි අද පාඩමට බහිමු. කලින් කීව්ව විදියට අද පාඩම encapsulation වුනාට java වල නියත පොඩි කොටසක් මං කලින්ම මතක් කරලා ඉන්නම්.. එතකොට අද පාඩම තේරුම් ගන්න එක ටිකක් ලේසි වෙයි.. ඒ නමා access modifies . මේ ගැන නම් කතා කරන්න දේකුත් නෑ.. මොකද අපි කලින් පාඩමකින් හොදටම කතා කලා මේක ගැන.. ඒ වුනාට පොඩි සාරාංශයක් කරලා දාන්නම් ...

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

ඔන්න ඕක නමයි access modifies ගැන කියන්න නියත සාරාංශ කතාව. තව දුරටත් අමතක වීමක් නියතවනම් මෙතනින් ගිහින් අපේ ඒ පාඩමත් බලන් එන්නකෝ...

හරි එහෙනම් අපි බලමු encapsulation කියන්නේ මොකක්ද කියලා. ඇත්තටම encapsulation වලින් කරන්නේ class එකක නියත data protect කරන එක. ඒ කියන්නේ class එක ඇතුළේ නියත attributes සහ methods ආරක්ෂා කරන එක. ගොඩක් වෙලාවට අපි protect කරන්න ගන්නේ attributes නමයි. . හරි එහෙනම් පුරුදු විදිහට මම උදාහරණයකින් පැහැදිලි කරන්නම්කො. මම ගන්නවා parson කියලා class එක. ඒකෙ attributes තුනක් ගන්නවා මම ගන්නවා name , age සහ city කියලා. හරි එහෙනම් අපි class එක හදාගෙන ඉමුකො.

```

class Person{

    String name;

    int age;

    String city;

}

```

හරි ඔන්න ඔය විදිහට නේද Person class එක එන්නේ. එහෙනම් අපි දැන් අපි parson ගෙ object එකකුත් හදාගෙන attributes වලට values assign කරමු.

```

class Person{

    String name;

    int age;

    String city;

}

class Demo{

    public static void main(String a[]){

        Person person1=new Person();
    }
}

```

```
    person1.name="john";

    person1.age=25;

    person1.city="Kandy";


    System.out.println("My name is :"+person1.name);

    System.out.println("My age is :"+person1.age);

    System.out.println("My city is :"+person1.city);

}

}
```

හරි output එකක් ලබන්නකෝ. කිසි අවුලක් නෑ නේද.



```
Command Prompt

C:\Users\USER PC\Desktop>javac Demo.java

C:\Users\USER PC\Desktop>java Demo
My name is :john
My age is :25
My city is :Kandy

C:\Users\USER PC\Desktop>
```

පොඩ්ඩක් එහෙනම් බලන්නකෝ ඔයාලා හදපු person class එක දිහා. ඒකෙ නියත attributes ඔක්කොටම කිසිම access modifier එක දාලා නෑ. ඒ කියන්නේ ඒ හැම attribute එකක්ම default attribute එකක්. default නම් තමන්ගේ package එකේ ඔක්කොටම access කරන්න පුළුවන් නේ. ඒ නිසා තමයි අපි එළියේ(Demo) class එකෙ ඉඳලා person class ඒකෙ attributes වලට values assign කරේ. හරි දැන් එහෙනම් හිතලා බලන්නකෝ ඔය attributes ඔක්කොටම private කළොත් මොකද වෙන්නේ කියලා. private attributes වලට access කරන්න පුළුවන් තමන්ගේ class එකේ අයට විතරයි නේද... එහෙනම් කොහොමද අපි Demo class එකෙ ඉඳලා මේ attributes වලට values assign කරන්නේ සහ ඒවා read කරන්නේ ?? අන්න එතකොට අපි මේ attributes වලට values assign කරන්න සහ ඒවා read කරන්න පුළුවන් වෙන විදියට method හදාගන්නවා. හරි දැන් කියන්න බලන්න මේ methods වල access modifier එක මොන වගේ එකක් වෙන්න ඕනෙද...method වල access modifier එක private වුනොත් වැඩක් නැහැ නේද.. මොකද අපි මේ methods හදා ගත්තේ අර privet attributes ටිකට access කරන්නනේ... එතකොට ඒ හදාගත්තු methods ටිකත් private වුනොත් ඒ methods ටිකටත් call කරන්න බැරි වෙනවා නේද.. අන්න ඒ නිසා අපි හදාගන්න methods ටික public උනානම් හරි. මොකද අපිට පුලුවන් එතකොට Demo class එකේ ඉඳදලා public methods වලට call කරලා අර privet attributes ටිකට values assign කරන්නයි ඒවා read කරන්නයි. මෙන්න මේ විදිහට private attributes වලට values assign හදන මේ .methodවලට අපි සාමාන්යයෙන් කියනවා getters සහ setters කියලා එහෙනම් අපි මේ getters සහ setters පාවිච්චි කරලා attributes වලට values assign කරන්නෙ කොහොමද කියලා බලමු.

```
class Person{

    privet String name;

    privet int age;

    privet String city;


    public void setName(String name){
```

```
        this.name=name;           // assign a name
    }

    public void setAge(int age){

        this.age=age;             // assign a age
    }

    public void setCity(String city){

        this.city=city;           // assign a sity
    }

    public void getName(){

        System.out.println("My name is :"+this.name); // getting the assigned n
ame
    }

    public void getAge(){

        System.out.println("My age is :"+this.age);    // getting the assigned ag
e
    }
```

```
public void getCity(){  
  
    System.out.println("My city is :"+this.city); // getting the assigned ci  
ty  
  
}  
  
}
```

```
class Demo{  
  
    public static void main(String a[]){  
  
  
        Person person1=new Person();  
  
        person1.setName("Smith");  
  
        person1.setAge(20);  
  
        person1.setCity("colombo");  
  
        person1.getName();  
  
        person1.getAge();  
  
        person1.getCity();  
  
  
    }
```


}

ඔබ්බ ඔය විදිහට තමයි getters සහ setters පාවිච්චි කරලා private attributes values sign කරන්නෙන් ඒව read කරන්නෙන්. this keyword එක පාවිච්චි කලේ ඇයි කියලා යාළුවෝ දන්නවනේ....අපි මේ කරපු දේ ගැන කෙටියෙන් කියනවා නම් , මෙයින් කරලා තියෙන්නේ class එකක තියන data වලට access කරන එක සීමා කිරීමයි. තවත් විදියකින් කියනවා නම් class data වලට access කරන එක සීමා කරලා data protect කිරීමයි . ඔබ්බ ඔය සරල ක්රියාවලියට තමයි encapsulation කියලා කියන්නේ. හරි අද පාඩම තේරුනා කියලා හිතනවා.

Abstract Class

ආයුබෝවන් ! oop පාඩම් මාලාවේ 11 වන දිගහැරුමත් සමග එකතු වෙන යාලුවෝ හැමෝම සාදරයෙන් පිළිගන්නවා...the sigma blog අඩවියත් සමග නිරන්තරයෙන් රැදී සිටි යාලුවෝ නම් මේ වෙනකොටත් object oriented programming ගැන ගොඩක් දේවල් ඉගෙනගෙන හමාරයි.ඔයා the sigma blog අඩවියට අලුත් සාමාජිකයෙක් නම් හරි , oop අවිල් කෙනෙක් නම් හරි අපේ මුල් පාඩම් ටික මෙතනින් ගිහින් බලන්න අමතක කරන්නන් එපා හොඳේ. හරි එහෙනත් පටන් ගමු අද පාඩම...අද පාඩම තමයි abstract class abstract class ගැන කතා කරන්න ඉස්සරින් කලින් පාඩම පොඩ්ඩක් මතක් කරන්නම්...කලින් පාඩමේදී අපි කතා කලා. encapsulation ගැන...මොකද්ද අපි encapsulation කියලා කිව්වේ... class එකක තියන data protect කරන එක එහෙමත් නැත්නම් data ආරක්ෂා කරන එකට අපි කිව්වා encapsulation කියලා.... අපේ encapsulation පාඩම මතක නැතිනම් මෙතනින් ගිහින් ඉක්මනට බලනකෝ...එතකොට සුපිරියටම මතක් වෙයි.

හරි එහෙනම් අපි දැන් බලමු අද පාඩම ගැන..අද පාඩම තමයි abstract class

abstract class එකක් ගැන කියන්න ඉස්සරින් මං කියන්නම් abstract method එකක් කියන්නේ මොකක්ද කියලා...ඇත්තටම abstract කියන වචනයෙන්ම කියවෙනවා නේද “මොකුත් නැ/හිස්” වගේ අදහසක්...ඒකම තමයි මේ class වලත් විශේෂත්වය...හැබැයි පොඩ්ඩක් වෙනස්...ඇත්තටම abstract

method එකක් කියන්නෙ body එකක් නැති එහෙමත් නැත්නම් meaningless
body එකක් නියන method එකකට...abstract method එකක් define
කරන්නේ method එක ඉදිරියෙන් abstract කියන වචනය දාලා...හරියට
නිකන් මෙන්න මේ වගේ...

```
class Person {  
  
    abstract void Jump(){  
  
    }  
  
}
```

ඔන්න ඔය විදියට නමයි abstract method එකක් define කරන්නේ...class
එකක් ඇතුලේ ඔය වගේ abstract method එකක් හරි ඊට වැඩි ගානක් හරි
නිබ්බොන් අන්න ඒ class එකත් abstract class එකක් වෙනවා... හරි
එහෙනම් Person class එක ඇතුලෙ නියනවා නේද jump කියලා abstract
method එකක් එහෙනම් අපේ Person class එකත් abstract වෙන්න ඕනි
නේද...

```
abstract class Person {  
  
    abstract void Jump(){  
  
    }  
  
}
```

```
}
```

ඔබ්බ ඔය වගේ class වලට තමයි abstract class එකක් කියන්නේ...abstract class මොකටද ගන්නේ කියලා අපි පසුව බලමු...දැනට මතක නියාගන්නකෝ abstract class එකක් කියන්නේ ඔබ්බ ඔකකට කියලා ☺☺..

මේ abstract class ගන්නොත් විශේෂතා ගොඩක් තියනවා....abstract method එකකනම් ඒකේ body එකේ මොකුත් නියා ගන්න බෑ ...නමුත් abstract class එකක් ගන්නොත් ඒකේ abstract methods තියෙන්නත් පුළුවන් abstract නොවන methods තියෙන්නත් පුළුවන්...ඔබ්බ ඔක එක විශේෂත්වයක් abstract class වල තියෙන...තවත් දෙයක් තමයි මේ abstract class වලට objects හදන්න බෑ....object හදන්න බැහැ කියන්නේ constructor එකකුත් නැහැ..ඔබ්බ ඔක තවත් විශේෂත්වයක්...පොඩ්ඩක් මේ උදාහරණය දිහා බලන්නකෝ...

```
abstract class Person{

    abstract void jump();

    void run(){

        system.out.println(" i can run fast !");

    }

}
```

```

}

class Student extends Person{

    void study(){

        system.out.println("i am studding !");

    }

}

```

හරි එහෙනම් ඔය class දෙක දිහා හොදට බලන්නකෝ...Person class එකෙන් Student class එක inherit වෙලා තියනවා .. එහෙනම් Person class එකේ තියන privet නොවන සියලුම attributes හා methods Student class එක ඇතුලට එනවා නේද...ඒ කියන්නේ Person class එකේ තියන jump කියන abstract class එකත් Student class එක ඇතුලට ඇවිත් තියෙන්නේ...කලින් මං කිව්වතේ class එකක් ඇතුලේ අඩුම එක abstract method එකක්වත් තිබ්බොත් class එක abstract වෙනවා කියලා....එහෙනම් අපේ Student class එකත් මෙන්න මේ වගේ abstract කරන්න ඕනි නේද...

```

abstract class Person{

    abstract void jump();
}

```

```
void run(){
```

```
    system.out.println(" i can run fast !");
```

```
}
```

```
}
```

```
abstract class Student extends Person{
```

```
    void study(){
```

```
        system.out.println("i am studding !");
```

```
    }
```

```
}
```

හරි ඔන්න ඔහොම තමයි අපේ code එක එන්නේ...කිසි අවිලක් නෑ...නමුත් මට ඕනි Student කෙනෙක්ගේ object එකක් හදන්න...Student class එක abstract එකක් නිසා මට Student කෙනෙක්ගේ object එකක් හදන්න compiler එක ඉඩ දෙන්නේ නෑ...එහෙනම් කොහොමද මං Student කෙනෙක්ගේ object එකක් හදන්නේ...Student class එකේ නියන abstract කැල්ල අයත් කරන්නම් හරි නේද ...එහෙම කරන්නම් abstract methods class එකෙන් අයිත් කරන්න ඕනි...Student Class එකේ නියන jump method එකෙන් abstract method එකකට නියෙන්නේ...අන්න ඒ jump method එකේ abstract බව අයිත් කලානම් හරි නේද...ඒ කියන්නේ jump method එක Student class එක ඇතුළෙදී override කලා නම් හරි නේද...override කලාම jump method එකට body එකක් නියන නිසා ඒ method එකේ abstract බව auto නැතිවෙලා යනවාcode එක ලිව්වොත් මෙන්න මේ වගේ

```
abstract class Person{

    abstract void jump();

    void run(){

        system.out.println(" i can run fast !");

    }

}
```

```

class Student extends Person{

    void jump(){

        system.out.println("i can jump !");           // overriding the abstract class

    }

    void study(){

        system.out.println("i am studding !");

    }

}

```

හරි දැන් මට අවිලක් නැතිව Student object එක හදන්න පුලුවන් නේද.....
 abstract class වල නියත තව විශේෂත්වයක් නමයි , abstract class reference
 variable එකකට පුලුවන් sub class objects store කර
 ගන්න..(upcasting)..අපේ උදාහරණයේ විදියටනම් මෙන්න මේ දේ...

```

Person person1= new Student();

```

හරි ඔන්න ඔය ටික තමයි abstract class ගැන කියන්න තියෙන්නේ...
interface වලදී abstract class හොදටම පාවිච්චි වෙනවා...එතකොට ඔයාලට
තවත් හොදට තේරෙයි...

හරි අපි abstract class ගැන කතා කලානේ...දැන් මං කතා කරන්න
හදන්නේ java වල තියෙන වැදගත් key word දෙකක් ගැන..කලින් අපි super
keyword එක ගැනයි this keywordඑක ගැනයි කතා කලානේ..අද කතා
කරන්න යන්නේ final keyword එක ගැනයි static keyword එක ගැනයි...

final keyword :

අැත්තටම කියනවනම් final keyword එකෙන් කරන්නේ variable එකකට
නිශ්චිත constant value එකක් assign කරන එක...උදාහරණයක් විදියට
ගන්නොත් වෘත්තයක ව.එ හොයද්දි අපට ඕනි වෙන $\pi = 3.14$ කියන
අගය...අපිට ඒක final keyword එක දාලා මේ විදියට define කෙරුව හැකි.

```
final float pi = 3.14 ;
```

ඔන්න ඕක තමයි final keyword එකෙන් කරන්නේ...

static keyword:

static keyword එකනම් ටිකක් වැදගත් keyword එකක්...අපිට static keyword
එක දාලා attributes define කරන්නත් පුළුවන් methods define කරන්නත්
පුළුවන්...attributes එකක් static කලොත් ඒ attribute එකේ value එක
program එක අවසන් වුනත් නොවෙනස්ව පවත්වා ගන්න පුළුවන්..හරියට
නිකන් මෙන්න මේ වගේ...

```
class Cat{  
  
    int age;  
  
    static int numOfSiblings=0;
```



```
void run(){
```

```
    System.out.print("it can be run fast !");
```

```
}
```

```
}
```

```
Class Demo{
```

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

```
        Cat cat1= new Cat(); // creating a cat object
```

```
        numOfSibligs ++;
```

```
        System.out.println("cat1 has "+ cat1.numOfSibligs +" sibligs");
```

```

        Cat cat2= new Cat(); //creating another object

        numOfSibligs ++ ;

        System.out.println("cat2 has "+ cat2.numOfSibligs +" sibligs");

    }

}

```

සාමාන්‍ය variable එකක නම් object එක එක පාරක් හැඳුවහම එහි අගය lost වෙලා යනවා. නමුත් static keyword එක පාවිච්චි කලොත් එහෙම වෙන්නෙ නැහැ. out put එකකවත් පොඩ්ඩක් බලන්නකෝ.

Out put :



```

C:\Users\USER PC\Desktop>javac Demo.java
C:\Users\USER PC\Desktop>java Demo
cat1 has 1 sibligs
cat2 has 2 sibligs
C:\Users\USER PC\Desktop>

```

ඔබ්බ ඔබ්බ attribute එකක් static කලාම වෙනත්...දැන් අපි බලමු method එකක් static කලාම මොකද වෙනත් කියලා...අන්තර්ගත method එකක් static කලාම , ඒ method එකට call කරන්න පුළුවන් object එකක් හදන්නේ නැතිව.ඒ කිව්වේ මෙන්න මේකයි.

```
class Cat{

    int age;

    static int numOfSiblings=0;

    static void run(){

        System.out.print("it can be run fast !");

    }

}

class Demo{
```

```
public static void main(String a[]){  
  
    Cat.run();  
  
}  
  
}
```

output :



```
Command Prompt  
C:\Users\USER PC\Desktop>java Demo  
It can be run fast !  
C:\Users\USER PC\Desktop>
```

ඔබ්න ඕක තමයි static keyword එකෙන් වෙන්නේ...

Interface

ආයුබෝවන් යාලුවෝ හැමෝටම...oop පාඩම් මාලාවේ අවසන් පාඩමට ඔයාලා හැමෝම සාදරයෙන් පිළිගන්නවා...ඔයාලා දිගටම අපිත් එක්ක රැදී හිටියනම් මේ වෙනකොටත් oop වල නියත ගොඩක් දේවල් ඉගෙන ගෙන හමාරයි...oop වල මොකක් හරි පාඩමක් මග හැරුනනම් **මෙතනින්** ගිහින් බලන්න පුළුවන්...oop වල අවසාන්ම පාඩමෙන් මං කතා කරන්න බලාපොරොත්තු වෙන්නේ interface ගැන...පුරුදු විදියටම මුලින්ම මම කලින් පාඩම පොඩ්ඩක් මතක් කරලා ඉන්නම්කෝ...කලින් පාඩමේදී අපි කතා කලා abstract classes ගැන...body එකක් නැති එහෙමත් නැත්නම් meaningless body එකක් නියත method එකක් හරි ඊට වැඩි ගානක් හරි නියත class එකකට අපි කිව්වා abstract class එකක් කියලා...තව abstract class ගැන ගොඩක් විස්තර කතා කලා කලින් පාඩමේ..කලින් පාඩමේ තිබුන හැම දෙයක්මනම් මතක් කරන්න යන්නේ නෑ...අමතක නම් හරි අපෙ කලින් පාඩම බලුවෙ නැත්නම් හරි මෙන්න **මෙතනින්** ගිහින් ඉක්මනට බලලා එන්නකෝ..

හරි එහෙනම් අපි දැන් බලමු අද පාඩම ගැන..අද පාඩම interface කියලා මම කලින් කිව්වනේ..හරි එහෙනන් අපි මුලින්ම බලමු interface එකක් කියන්නේ මොකද්ද කියලා...interface කියන්නේ මොකද්ද කියලා කියන්න කලින් මං කියන්නම් ඇයි interface පාවිච්චි කරන්නේ කියලා....හරි අපි කලින් පාඩමකින් කතා කලානේ **inheritance** ගැන...එතනදී අපි කලේ එක class එකකින් තවත් class. එකක් inherit කරපු එකනේ...ඒ කියන්නේ එක inheritance එකයි තිබුනේ...multiple inheritance ගැන අපි කතා කලේ වත් නෑනේ නේද...එහෙම කලේ multiple inheritance java වල නැති නිසා...(C++ වල නම් නියනවා) ...multiple inheritance කියන්නේ මේ වගේ එකකට...

```
class A{  
  
    // statement  
  
}
```

```

class B{

    // statement

}

class C extends A,B{

    // statement

}

```

ඔත්ත ඕකට තමයි multiple inheritance කියලා කියන්නේ...**එක class එකක් class ගොඩකින් inherit වෙන එකට**...ඔත්ත ඔය දේ කරන්න java language එකේ ඉඩදෙන්නේ නෑ...අන්න ඒ දුර්වලතාව මගහැර ගන්න තමයි interface පාවිච්චි කරන්නේ...interface එක හැසිරීම class එකකට ගොඩක් දුරට සමානයි...නමුත් class එකක් නම් නෙවෙයි...interface වලින් පුලුවන් class කිහිපයක නියත ලක්ෂණ එක class එකකට ගන්න...හරියට නිකන් inheritance පාවිච්චි කලා වගේ..abstract class ගැන කතා කලා වගේම interface වල නියත ලක්ෂණ ටිකක් මුලින්ම මං කියන්නම්...

interface එකක් අපි define කරන්නේ interface කියලා දාලා interface එකේ නම ලියනවා..මෙන්න මේ විදියට..

```

interface myInterface{

```

```
// statement
```

```
}
```

ඔබ්බ ඔහොමයි interface එකක් define කරන්නේ...

interface එකක නියෝග තවත් විශේෂත්වයක් නම් interface එකක නියා ගන්න පුළුවන් abstract methods විතරයි.. interface එකක් ඇතුළේ නියා ගන්නේ abstract method විතරයිනම් interface එකත් abstract වෙන්න ඕනි නේද....interface එක abstract වුනා කියන්නේ object හදන්නත් බැහැ ...object හදන්න බැ කියන්නේ constructor එකකුත් නෑ නේද.. තව දෙයක් නියනවා , interface එකක් ඇතුළේ නියන methods ඔක්කොම public වෙන්නත් ඕනි abstract වෙන්නත් ඕනි static වෙන්නත් ඕනි...ඒ වගේම interface එක ඇතුළේ නියන attributes ඔක්කොම public වෙන්නත් ඕනි static වෙන්නත් ඕනි final වෙන්නත් ඕනි..තවත් දෙයක් නියනවා..intercace එකක reference variable එකකට පුළුවන් implements වුන class එකේ objects store කරගන්නත්..හරියට නිකත් මෙන්න මේ වගේ..

```
interface A{
```

```
    // statement
```

```
}
```

```
class B implements A{
```

```
    // statement
```

```
}
```

```

class Demo {

    public static void main(String a[]){

        interface A=new B() // upcasting

    }

}

```

ඕනෑම ඔය ටික තමයි interface එකක නියත ජර්ධන විශේෂත්ව...

හරි දැන් අපි බලමු interface පාවිච්චි කරන්නේ කොහොමද කියලා...කලින් මං කිව්වනේ interface කියන්නේ inheritance වගේ කියලා...inheritance වලදී නම් එක class එකකින් තවත් class එකක් inherit කරන්නේ extends keyword එක පාවිච්චි කරලා...interface එකක් නම් implements keyword එක දාලා තමයි class එකට සම්බන්ධ කරන්නේ...මේ විදියට interface ඕනි ගානකින් එක class එකක් implements කරන්න පුලුවන්..

```

interface A{

    abstract void a();           // abstract method of interface A..

}

```



```
interface B{

    abstract void b();        // abstract method of interface B..

}
```

```
class X{

    int x;

    void a(){

        // statement ;        // not an abstract method

    }

}
```

```
class Demo extends X implements A,B{
```

```
// statements;
```

```
}
```

Demo class එක X ගෙන් inherit කරලා තියනවා, A හා B ගෙන් implements කරලා තියනවා...හරි ඔන්න ඔය විදියට තමයි class එකක් interface එකකින් implements කරන්නේ..හරි තේරුනා නේද වැඩේ....interface ගැන ඉතිං ඔච්චරයි කියන්න තියෙන්නේ☺☺☺...තේරුනා නේද අද පාඩම....ඔන්න ඔය පාඩමත් සමගම අපේ oop පාඩම් මාලාවත් නිමාවෙනවා...oop වල මොකක් හරි අවිලක් ආවොත් ඉක්මනට comment එකක්