# **java need to know theories**

## child classes and parent classes (Inheritence)

අපිට game එකක් develop කිරීමට ඇතැයි සිතන්න.එම game එකේ බොහෝ සතුන් ඇතැයි සිතන්න.එසේ නැත්තම් සත්තු පමණක් ඉන්න game එකක් ලෙස සිතමු.එවිට ඒ සෑම සතෙක් සදහාම class එකක් තිබීම හොදයි.මොකද එවිට එම class එක භාවිතා කරලා අපිට එක class වර්ගයක objects විශාල ප්‍රමයක් නිපදවීමට පුලවන් නිසා.එනම් elephant නම් class එක භාවිතා කරලා elephant objects විශාල ප්‍රමයක් සැදීමට පුළුවන්.මෙනිසා අපිට game එකේ elephants ඉන්නවනම් 20ක් පමණ,ඒ විස්ස සදහාම වෙන වෙනම code කිරීමේ අවශ්‍ය තාව නැති වෙනවා.elephant class එකේ objects 20ක් සෑදීම පමණයි කල යුත්තේ.එම නිසා විශාල ලෙස code කිරීමේ අවශ්‍ය තාව මග හරවා ගැනීමට,කාලය ඉතුරු කර ගැනීම සදහා,errors අඩු කරගැනීම සදහා, සහ එවැනි තවත් ගොඩක් වාසි ගෙන දෙන ක්‍රමයක් තමා class නිපදවා, එම class වල objects සෑදීමේ ක්‍රමය.එනම් object oriented approach එක ගොඩක් සරලයි සහ ඉහත කියන ලද වාසි ගෙන දෙනවා code කරද්දී.

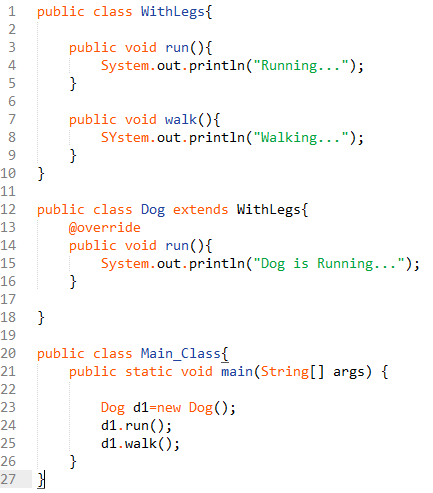
අපිට තවත් තර්කානුකුලව සිතන්න පුලවන්.එනම් අපි දන්නවා සතුන්ව බෙදා වෙන් කරන්න පුළුවන් ඇවිදින සත්තු(එනම් කකුල් තියෙන සත්තු),ඉගිලෙන සත්තු,බඩගාන සත්තු සහ පීනන සත්තු ලෙස.ඉතින් අපිට මෙහෙම හිතන්න පුලවන්, කකුල් තියෙන සත්තු හැමෝම ඇවිදිනවා,පියාබන සත්තු හැමදෙනාම පියාබනව, මෙලස යම් යම් සතුන්ගේ පොදු ලක්ෂණ,පොදු ක්‍රියාකාරකම් තියෙනවා.ඉතින් හැම සතෙක්ගේම class එකේ, එම පොදු ක්‍රියාකරකම් වෙන වෙනම හැම class එකකම code කරනවට වඩා ලේසි නැද්ද,මොකක් හරි ක්‍රමයකින් අපිට කියන්ඩ පුළුවන් නම් “ඔයාල මෙන්න මේ සත්ව කණ්ඩායමට අයත් කට්ටියක්,එම නිසා එම සත්ව කණ්ඩායමට අයත් පොදු ලක්ෂණ සහ ක්‍රියාකාරම් ඔයාලටත් තියෙනවා” කියල, එය ඉතා පහසු වෙනවා,සහ කාලය ඉතුරු කරනවා හැම class එකකම වෙන වෙනම ඒ class එකේ වර්ගය අනුව, එකම walking(),runing(),flying(),swim() වැනි methods වල code ලියනවට වඩා.

මෙසේ කිරීමට ක්‍රම තියෙනවා object oriented programming වල.එනම් අපිට පුළුවන් පොදුවේ තියෙන methods තියෙන classes හදන්න ඒ ඒ group සදහා වෙන වෙනම.එනම් අපිට class එකක් හදන්න පුලවන් WithLegs කියලා.මෙහි අපිට walk(),run(),runFast() වැනි methods හදන්න පුලවන්.ඉන් පසු අපිට පුලවන් කකුල් තියෙන animal කෙනෙක්ගේ class එකකට මේ ඉහත කියන ලද class එක extend කරන්ඩ කියලා (එහෙම නැත්තම් WithLegs කියල class එකක් නැතුව Interface එකක් හදලා,එය implement කරන්න කියලත් කියන්න පුලවන් ඕනනම්, මේවා ගැන පසුව හරියටම පහල පැහැදිලි කරලා තියෙනවා)

ඉහත කිව්වේ object oriented language වල තියෙන ප්‍රධානම ලක්ෂණ 4න් එකක් වන inheritance කියන දෙය ගැනය.එනම් අපිට වෙන class එකක තියෙන methods වැනි දෑ උරුම කර ගන්න පුලවන්.එනම් WithLegs class එක parent class එකක් ලෙස සිතලා සහ කකුල් තියෙන සත්තුන්ගේ classes child classes කියල හිතන්න පුළුවන්. extend යන වචනයෙන් කියන්නේ parent class එකේ methods සහ varaibles ඔයාටත් උරුමයි කියලයි. එම නිසා එම methods child class එක තුලද භාවිතා කල හැක.

එසේම parent class එකේ private නොමැති සියලු methods සහ attribute child class වලට උරුම වෙන නිසා එම parent class එකේ object එකක් හදලා එහි methods call කල යුතු නෑ.නිකන්ම තමන්ගේ methods වගේම call කරන්න පුලවන් child class object වලට.එකයි උරුම වෙනවා කියන්නේ.නැත්තම් වෙන class එකක methods access කරන්න තවත් class එකක සිට අනිවාර්යෙන් එම method තියෙන class එකේ object එකක් හදලා තිබිය යුතුයි එම class එකේ සිට.

පහත උදාහරණය බලපුහාම ඉහත කියන ලද දෙය පැහැදිලිව බලා ගැනීමට පුළුවනි.



output එක

Dog is Running...

Walking...

මේ අනුව පැහැදිලියි අපිට d1 කියන reference variable එක භාවිතා කරලා walk() යන method එකත් call කරන්න පුළුවන්.walk() කියල method එකක් Dog class එක තුල ලියලාවත් නෑ.නමුත් Dog class එක WithLegs යන class එක extend කරන නිසා (inherit කරන නිසා) WithLegs යන class එකේ තියෙන methods සියල්ල තමන්ගේ වගේ භාවිතා කල හැකි.තවද අපිට පේනවා Dog class එක තුලදී run() යන එයට inherit වන method එක තමන්ට කැමති පරදී වෙනස් කරගෙන තියෙනවා කියල.එයට කියනවා overriding කියල.

# Interfaces

interface එකක් යනු abstract methods සහ constants පමණක් තියෙන class එකක් ලෙස සිතුව හැකියි.යම් child class එකක්, parent class එකක් හෝ abstract class එකක් extend කරනවා කියලයි අපි කියන්නේ සහ code කරන්නේ.නමුත් අපි interface සමග භාවිතා කරන්නේ extend යන වචනය වෙනුවට implement යන වචනයයි.එනම් යම් class එකක් implement කරනවා යම් interface එකක් යනුවෙනි.implement යන්නෙහි අදහස ක්‍රියාත්මක කරනවා,සම්පුර්ණ කරනවා වැනි දෙයකි.එනම් යම් class එකක් interface එකක් implement කළහොත්, එම interface එක තියෙන සියලු abstract methods මෙම class එකේදී ක්‍රියාත්මක කිරීම එහෙම නැත්තම් එම abstract methods සියල්ල සම්පුර්ණ කරීම අනිවාර්යයි.එසේ එම abstract methods සම්පුර්ණ කිරීමටත් අපි කියන්නේ implement කරනවා කියලයි.

Interface මගින් බොහෝ දුරට programmerට මතක් කිරීමේ කාර්ය කල හැක.එනම් යම් class එකක තිබිය යුතුම methods අනිවාර්යෙන් implement (define) කිරීමට programmer ව පොලබවනු ලබනවා.මොකද Interface එකක ඇති abstract methods සියල්ල එම Interface එක implement කරන ඕනෑම class එකක් මගින් අනිවාර්යෙන් භාවිතා කල යුතුමයි,නැත්තම් compilation errors එනවා.

implementation එක ගැන දන්නේම නැත්තම්,එනම් class ටිකක් අතර යම් පොදු method එකක් තියෙන්ඩ ඕන කියලත් දන්නවා යම් නමුත්, ඒ ඒ class වලදී එම method එක ක්‍රියාත්මක කරනවන ආකාරය හොදටම වෙනස් නම්, හෝ ඒ ඒ class වලදී එම පොදු method එක ක්‍රියාත්මක කරවන ආකාරය පිළිබද කිසි අවබෝදයක් අපිට නැත්තම්, එවැනි methods එකතු කර Interface එකක් සැදීමයි කල යුත්තේ.

# abstract classes

# Local classes

කිසියම් හෝ block එකක් තුල define කර ඇති classes වලට Local classes කියනු ලැබේ.මෙහිදී block එකක් යනු method එකක්,loop එකක්,if clause එකක් වැනි ඕනෑම දෙයක් විය හැකි.නමුත් බොහෝ වෙලාවට method එකක් තුල මෙම Local classes නිපදවනවා දැකගන්න පුළුවන් අපිට.

උදා:

public class A{

public static void method(){

class B{

public void methodB{

system.out.print(“hi”);

}

}

}

}

මෙහි B නම් class එක local class එකක් වේ.තවද මෙම class එක define කරල තියෙන්නේ static method එකක් තුලයි.එම නිසා B නම් local class එකට, පිට class එකෙන් access කල හැක්කේ static දේවල් පමණයි.(local classes සැදිය යුත්තේ static methods තුල පමණයි කියල දෙයක් නෑ).

සාමාන්‍යයෙන් local classes සලකන්නේ non-static දේවල් ලෙසයි.එම නිසා ඒවාට access කල හැක්කේ එයට පිටින් තියෙන block එකේ instance members (non static දේවල්) පමණයි.එලසම local classes non-static ලෙස සලකන නිසා එම class එක තුලද බොහෝ වේලාවන් වලදී static ලෙස දේවල් declare කරන්න දෙන්නේ නෑ java වලදී.

# Nested Classes

java වලදී අපිට class එකක් තුල තවත් class එකක් සැදීමට අවස්ථාව සලසනවා.එලස ඇතුලේ සාදන classes වලට nested class යැයි කියනු ලැබේ.

local class එකක් යනු යම් class එකක් තුල ඇති method එකක් තුල හදන class එකක් වේ.නමුත් Nested classes යනු එලස method එකක්(block එකක්) තුල නැතුව නිකන්ම තවත් class එකක් තුල හදන classes වලට වේ.

උදා:

class OuterClass{

...

class NestedClass{

...

}

}

nested classes කොටස් දෙකකට බෙදා වෙන් කල හැක.එනම් static ලෙස හා non-static ලෙසයි.static ලෙස declare කරන nested class වලට අපි static nested classes ලෙස කියනවා.non-static ලෙස declare කරන nested class වලට අපි inner classes යැයි කියනවා.

class OuterClass{

...

static class StaticNestedClass{

...

}

class InnerClass{

...

}

}

nested class එකක් එයට පිටිතින් තියෙන class එකේ member කෙනෙක් කියල සිතුවා හැකයි.(class එකට අයිති variables,methods වටද අපි පොදුවේ කියන්නේ class එකේ members ල කියලයි.).Non-static nested classes (inner classes) වලට පිට class එකේ members ලව ඒවා private වුනත් access කල හැක.static nested classes වලට enclosing class එකේ (පිට class එකේ) members ලව access කල නොහැක.enclosing class එකේ member කෙනෙක් යන කරුණ සලකා,ඕනෑම nested class එකක් private,public,protected හෝ package private ලෙස declare කල හැක.(outer classes declare කල හැක්කේ public ලෙස හෝ package එක සදහා private ලෙස පමණි)

## Why Use Nested Classes?

යම් class එකක එක object එකක් පමණක් තවත් class එකකට අවශ්‍ය විට අපි එම object එකේ class එක අනෙක් class එක තුල code කිරීම සිදු කරයි.

inner class එකක් ලෙස සැදුව විට nested class එකට enclosing class එකේ members පවා access කිරීමට හැකි වීම.so සාමාන්‍ය ලෙස object සාදා,එමගින් members access කිරීම කල යුතු නෑ.

යම් class එකක private ලෙස define කර ඇති කිසිම member එකක් එහි child classes වලටවත් access කල නොහැක.මෙලස යම් class එකක private members වෙනත් class එකෙක් මගින් access කිරීමට අවශ්‍ය වන විට එම class එක nested class එකක් ලෙස code කරානම් හරි.මොකද nested class වලට enclosing class එකේ private members පවා access කල හැක.

# Anonymus Classes

යම් class එකක් declare කරන අවස්ථාවේම එය instantiate කල යුතුයි නම් (එහි object එකක් සැදිය යුතුයි නම්) අපිට එම class එක anonymous class එකක් ලෙස code කල හැක.ඒවා local classes වගේම තමා,එකම වෙනස ඒවාට නමක් නැති වීමයි(එම නිසයි anonymous කියල කියන්නේ).යම් local class එකක් එක වරක් පමණයි නම් අවශ්‍ය වනේ (එහි එක object එකක් පමණයි නම් අවශ්‍ය වන්නේ) එවිට මෙම anonymous class ක්‍රමය භාවිතා කරන්න.

## Declaring anonymous classes

local classes යනු class declaration එකක් (නියම කිරීමක්,ප්‍රසිද්ධ කිරීමක්) ලෙස සිතුව හැකියි.මෙලසම anonymous classes යනු expressions ලෙස සිතුව හැකියි.මොකද අපි class එක declare කරනවා පමණක් නෙමේ, object එකක් සැදීමද කල යුතුයි.object සැදීම expression එකක් ලෙසයි සමනයයෙනුත් අපි code කරන්නේ.so anonymous class එකක් අපි සැලකිය යුත්තේ expression එකක් ලෙසයි.

මෙනිසා new යන විදානය සමග constructor එකක් අවශ්‍ය වෙනවා object එක සැදීම සදහා.නමුත් constructor එක ගන්න අපි තවම class එක define කරලා නැහැ.අනික class එකට නමක් නැති නිසා මෙයට constructor එකක් කියල දෙයක් නොමැත.එසේනම් අපි constructor method එක ලෙස ගන්නේ කුමක්ද?

අපි දන්නවා anonymous class යනු nested class වර්ගයක්ම තමා.හරියටම කියනවනම් local class එකක් වගේ.එනම් method එකක් ඇතුලේ,if claus එකක් ඇතුලේ වැනි තැනකයි මෙම anonymous classes define කරන්නේ.so මෙම හදන object එක refere කිරම සදහා reference variable එකක්ද අවශ්‍යයි.මෙම class එක anonymous නිසා (නමක් නෑ,එම නිසා type එකක් නෑ මෙම class එක නමින්) එම object එක refer කරන variable එකේ type එක ලෙස ගන්නේ කුමක්ද?

anonymous classes හදන්නෙම එක්කෝ class එකක් පවතින class එකක් extend කරලා එහෙම නැත්තම් interface එකක් implement කිරීමෙන්ය.නමුත් මෙහිදී class එකේ නමක් නැති නිසා, extend කිරීමට හෝ implement කිරීමට extends යන වචනය හෝ implements යන වචනය භාවිතා නොකරයි.එසේ භාවිතා නොකරත්,සැබවින්ම වෙන්නේ එයයි.

එම නිසා new යන command එක සමග දමන constructor method එක වන්නේ මෙම extend කරන class එකේ constructor එක හෝ implement කරන interface එකේ constructor එකයි.එලසම එවිට සෑදෙන object එකේ reference variable එකේ type එක ලෙසද extendකරන class එකේ type එක හෝ implement කරන interface එකේ type එක ගත හැක.එම නිසා anonymous class එකක් define කරන basic syntax එක පහත පරිදි වේ.

public class Outer{

public void meth(){

IntrfaceName refrnce =new IntrfaceName(){

.... //anonym class variables

....// anonym class methods()

}**;**

}

}

මෙම semi colan එක අනිවාර්යයි.මොකද anonymous class එකක් යනු statement එකකි.