

මූලික Java පාඨම :: Java හැඳින්වීම

by Kaniskha Dilshan

ජාවා පරිගණක භාෂාව භාවිතා කරන විවිධ පරිගණක මෙහෙයුම් පද්ධති වල එකසේ ක්‍රියාත්මක වන මෘදුකාංග නිපදවීමට පුළුවන්. ජංගම දුරකථන, ඩෙක්ස්ටොප් පරිගණක මෘදුකාංග සහ විශාල ජ්‍යෙෂ්ඨතායෙන් ප්‍රමාණයක් භාවිතා කරන සර්වර් සයිඩ් භාවිත යෙදුම් නිපදවීමටද ජාවා පරිගණක භාෂාව උපකාරී කරගන්නා පුළුවන්.

C/C++ වගේ නොවෙයි ජාවා පරිගණක භාෂාව interpreted පරිගණක භාෂාවක්. ඒ කියන්නේ ජාවා වැඩසටහනක තිබෙන්නේ මැෂින් ඉන්ස්ට්‍රක්ෂන්ස් නොවෙයි, ජාවා වර්දුවල් මැෂින් එකට හඳුනාගන්නා පුළුවන් ඉන්ස්ට්‍රක්ෂන්ස්. ජාවා වර්දුවල් මැෂින් එක තමයි ජාවා වැඩසටහන් ධාවනය කරවන්නේ. එතකොට ජාවා වල එක අවසියස් තමයි C/C++ තරම් කාර්යක්ෂමතාවයක් ලබා නොදීම. ඒ වුනාට ජාවා වල තිබෙන වාසිත් එක්ක බලනකොට අවාසි නොසලකා හැරිය හැකි තරම්. කාර්යක්ෂමතාවය පිළිබඳ ගැටලුව උනත් නිරාකරණය කරගන්න පහසුකම් ජාවා තුළින්ම සොයාගන්න පුළුවන්.

ජාවා පරිගණක භාෂාව වස්තු පාදක OO(Object Oriented) පරිගණක භාෂාවක්. මේ නිසා මනාකොට සංවිධානය කරන ලද විශාල ප්‍රමාණයේ පරිගණක වැඩ සටහන් ලිවීමට ජාවා යොදාගන්න පුළුවන්.

Java Applications හා Java Applets අතර ඇති වෙනස්කම්

Sunday, April 4, 2010 by Kaniskha Dilshan

Java Applets

- ධාවනය වීමට ජාවා සඳහා සහය දක්වන වෙබ් බ්‍රවුසරයක් අවශ්‍ය වේ.
- සීමා කරන ලද පරිසරයක් තුළ ධාවනය වේ (Sand box).
- ධාවනය වන පරිගණකයේ ඇති බොහෝ සම්පත් (File system, Networks..etc) භාවිතා කළ නොහැකි වුනත් එය හොඳට කර . ඇති සර්වර් පරිගණකයේ සම්පත් භාවිතා කළ හැක.
- ධාවනය වීමට පෙර අන්තර්ජාලය හරහා පරිගණකයට භාගත කළ යුතුය යව සිදුකරනවුසරය ස්වයංක්‍රීයවම වෙබ් බ්‍ර : සැයු . ක්‍රියාවලියකියෙන් සාමාන්‍ය . පරිශීලකයා මෙයට මැදිහත් විය යුතු නොවේ applet ටැගය මගින් වෙබ් බ්‍රවුසරය ජාවා applet එක හඳුනා ගනී(.

Java Applications

- ජාවාහි උපරිම හැකියාව ලබාගත හැකිය
- ධාවනය වන පරිගණකයේ සම්පත් මුළුමනින්ම පාහේ භාවිතා කළ හැක.
- සාමාන්‍යයෙන් ස්ථාපනය කිරීමකින් අනතුරුව ධාවනය (run) කළ හැක.

සැයු අප ඉදිරි පාඨම වලදී වැඩි දුරටත් ජාවා :applets සහ applications පිළිබඳ හැදෑරීමට බලාපොරොත්තු වේ.

ජාවා තුළ භාවිතාවන මූලික දත්ත ආකාර (Basic Data Types of Java)

Wednesday, April 7, 2010 by Kaniskha Dilshan

ජාවා යනු විචල්‍යයන්හි(Variables) දත්ත ආකාරය පිළිබඳ තදින් සැලකිලිමත් වන පරිගණක භාෂාවකි (Strongly typed). එනම් අපි ජාවා තුළ විචල්‍යයන් අර්ථ දක්වන විට අනිවාර්යයෙන්ම එම විචල්‍යය කුමන ආකාරයේ දත්ත රැඳවීමට භාවිතා කරන්නේද යන්න සඳහන් කළ යුතුය කුමනවිචල්‍ය . ආකාරයේ දත්ත රැඳවීමට භාවිතා කරන්නේද යන්න සඳහන් කළ යුතුය ඊට අමතරව . විචල්‍යයන් භාවිතා වන විටදීද දත්ත ආකාරය ගැන තදින් සැලකිලිමත් විය යුතුය. නැතිනම් ජාවා සම්පාදකය (compiler) ඒ බව දන්වමින් දෝශ පනිවුඩයක් (syntax error) ලබා දෙනු ඇති.

අපි දැන් බලමු ජාවා තුළ භාවිතාවන දත්ත ආකාර පිළිබඳ විස්තරයක්

දත්ත ආකාරය (Data Type)	මූලපදය (Keyword)	රඳවාගත හැකි අවම අගය (Min Value)	රඳවාගත හැකි උපරිම අගය (Max Value)	ප්‍රාරම්භක අගය (Initial Value)
Byte	byte	-128	127	0
Float	float	1.4E-45	3.4028235E38	0.0F

Double	double	4.9E-324	1.7976931348623157E308	0.0D
Integer	int	-2147483648	2147483647	0
Long	long	-9223372036854775808	9223372036854775807	0L
Short	short	-32768	32767	0
Character	char	'\u0000'	'\uffff'	'\u0000'

ජාවාහි නිතර භාවිතා වන දත්ත ආකාරයක් වන String දත්ත ආකාරය ඉහත වගුවේ සඳහන් ආකාරයේ දත්ත ආකාර මෙන් මූලික දත්ත ආකාරයක් නොවේ එය .Class එකකි . අපිString විචල්‍යයක් සාදන විට සිදුවන්නේ String object එකක් සෑදීමය) රම්භක අගයහි ප්‍රා .Initial Value) වනුයේ null ය .String අපි classes හා objects පිළිබඳ ඉදිරි පාඩම පෙළකදී විස්තරාත්මක ලෙස බලමු.

අපි දැන් බලමු ජාවා වැඩසටහනක් තුළ විචල්‍යයන් භාවිතාවන්නේ කොහොමද කියල.

විචල්‍යයන් අර්ථ දක්වන අයුරු (Defining a Variable)

Syntax : `datatype variable_name`

```
1: int number;
2: float temperature;
3: String name;
4: char first_letter;
```

විචල්‍යයන් සඳහා අගයන් ආදේශ කරන අයුරු (Assigning values)

Syntax : `variable_name = value`

```
1: number=1133;
2: temperature=33.23;
3: name="Kanishka Dilshan";
4: first_letter='K'
```

මෙහිදී අගයන් ආදේශ කිරීමේදී String අගයන් සඳහා ද්වි උද්ධෘත පාඨයන් ද char අගයන් සඳහා තනි උද්ධෘත පාඨයන්ද භාවිතා කළ යුතුය.

අවශ්‍ය නම් විචල්‍යය අර්ථ දක්වන විටම වුවද අගයන් ආදේශ කිරීම කළ හැක.

උදා : `int number=1133;`

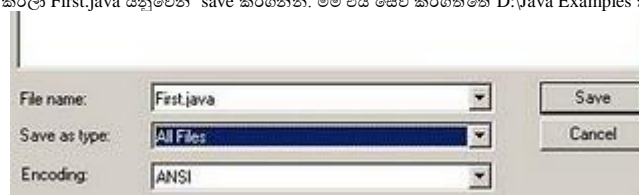
අපි සරල ජාවා වැඩසටහනක් ලියමු

Thursday, April 8, 2010 by Kanishka Dilshan

අපි මීට පෙර පාඩමෙන් ජාවා Variables පිළිබඳ කතාකලා.අපි අද පාඩමේදී සරල ජාවා වැඩසටහනක් ලියන ආකාරය බලමු. ජාවා කියන්නේ වස්තු පාදක (Object Oriented) පරිගණක භාෂාවක් කියලා දන්නවානේ. සැම දෙයක්ම Object එකක් කියලා තමයි ජාවා තුලදී සැලකෙන්නේ. කොටින්ම කිව්වොත් අපි ලියන ජාවා වැඩසටහන පවා ජාවා තුළ සැලකෙන්නේ Object එකක් ලෙසටයි. අපි දැන් බලමු ජාවා වැඩසටහනක සරලම අවස්ථාවක්.

```
1. public class First {
2.     public static void main(String args[]){
3.         System.out.println("Hello Sri Lanka");
4.     }
5. }
```

මෙම කෝඩ එක Notepad එකේ ටයිප් කරලා First.java යනුවෙන් save කරගන්න. මම එය සේවි කරගත්තේ D:\Java Examples කියන ෆෝල්ඩර් එකේ.



දැන් කියෙන්නේ ජාවා වැඩසටහන සම්පාදනය කිරීමයි (compile). මේ සඳහා ඔබේ පරිගණකයේ Java Development Kit ස්ථාපනය කර තිබිය යුතුයි. මම මීට පෙර පාඩමකදී ජාවා ස්ථාපනය කරගන්නා අයුරු ඔබට පෙන්වාදී තිබෙනවා. එම පාඩම බැලීමට [මෙතන ක්ලික් කරන්න.](#)

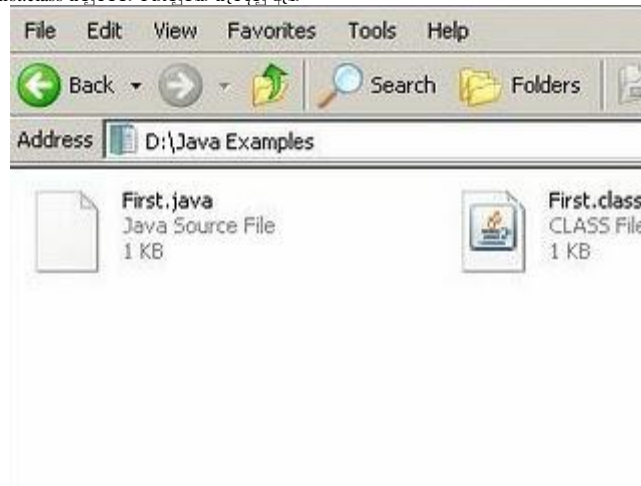
වැඩසටහන කම්පයිල් කරගැනීමට Command Prompt හි පහත විධානය ලබා දෙන්න.
javac First.java
ප්‍රතිඵලය :

```
C:\> Command Prompt

D:\Java Examples>javac First.java

D:\Java Examples>_
```

සාර්ථකව කම්පයිල් වූයේ නැත්නම් නැවත වැඩසටහන පරීක්ෂා කර බලන්න. ජාවා Case Sensitive නිසා Capital, Simple නිවැරදිව තිබිය යුතුය. සාර්ථකව කම්පයිල් වූ විට පහත පරිදි First.java වලට අමතරව First.class යනුවෙන් ගොනුවක් සෑදෙනු ඇතී



දැන් තිබෙන්නේ කම්පයිල් වූ වැඩසටහන ධාවනය කිරීමයි (Run). මේ සඳහා පහත විධානය භාවිත කරන්න.
java First
ප්‍රතිඵලය :

```
C:\> Command Prompt

D:\Java Examples>java First
Hello Sri Lanka

D:\Java Examples>_
```

අපි දැන් ලියන ලද ජාවා වැඩසටහන ගැන වැඩිදුරටත් අධ්‍යයනයක යෙදෙමු.

Line 1 : මෙම ලයින එකෙන් අප ලියන ලද ජාවා වැඩසටහන අයත් class එක සඳහන් කිරීම සිදුකෙරෙනවා. ජාවා ගොනුව save කරගත යුත්තේ මෙම නමිනුයි. අපේ වැඩසටහනේ එය First ලෙස දී තිබෙනවා

Line 2: මෙමගින් අප ජාවා වැඩසටහනේ ප්‍රධාන method එකෙහි ආරම්භය සනිටුහන් වනවා. මෙම method එක හරහා තමයි අනෙකුත් සියලුම methods call කිරීම සිදුවන්නේ.

Line 3: මෙහිදී println() නම් method එක call කිරීම සිදුකෙරෙනවා. එය ඇතුළුවන්නේ System නම් package එක තුලයි. out යනු ජාවාහි standard output stream එකයි(අපි ඉදිරි පාඩම් පෙලකදී ජාවාහි streams පිළිබඳ වැඩිදුරටත් හදාරමු). මෙය සාමාන්‍යයෙන් Command Prompt එකයි. මෙම සමස්ථ ජෙලිය මගින් සිදුකරන්නේ Hello Sri Lanka යනුවෙන් command prompt / console එකෙහි පෙන්වීමයි.

අපි ඊළඟ පාඩමෙන් ජාවා operators පිළිබඳව බලමු.

ජාවා කාරක (Java Operators) 1 කොටස

Friday, April 9, 2010 by Kaniskha Dilshan

කලින් පාඩමෙන් පොරොන්දු පරිදිම මෙම පාඩමෙන් ජාවා operators පිළිබඳ සාකච්චා කරන්න මම හිතුවා. කාරක(operators) යනු ඕනෑම පරිගණක භාෂාවක අනිවාර්ය කොටසක්, මොකද ඔපරේටර් නොමැතිව අංක ගණිතමය, චිජ් ගණිතමය හෝ තාර්කික අවශ්‍යතා ඉටුකරගත නොහැකියි. ජාවා තුල භාවිතාවන ඔපරේටර් සියල්ලක්ම පාහේ භාවිතා වන ආකාරය C/C++ වලදී ඔපරේටර් භාවිතා වන ආකාරයට සමානයි. අපි දැන් බලමු ජාවා තුල භාවිතා වන මූලික ඔපරේටර් වර්ග මොනවාද කියලා.

1. Arithmetic operators
2. Relational operators
3. Boolean logical operators
4. Bitwise operators

අපි මෙම පාඩමෙන් සාකච්ඡා කරන්නේ Arithmetic operators පිළිබඳවයි.

මෙහිදී අපි a හා b යනුවෙන් integer විචල්‍යයන් 2 ක් යොදාගනිමු. a=14 ලෙසද b=8 ලෙසද assign කරමු. ප්‍රථමය රඳවා ගැනීමට c යනුවෙන් integer variable එකක්ද ගනිමු. පහත ඉදිරිපත් කර ඇති වගුව භාවිතයෙන් මේවායේ ක්‍රියාකාරීත්වය තේරුම් ගැනීමට උත්සාහ කරමු.
int a=14; int b=8; int c;

කාරකය (operator)	කාරකයේ නම operator name	භාවිතය usage	ප්‍රතිඵලය result
+	Addition	c=a+b	c=22
-	Subtraction	c=a-b	c=6
*	Multiplication	c=a*b	c=112
/	Division	c=a/b	c=1
%	Modulus	c=a%b	c=6
+=	Addition Assignment	a+=b	a=22
-=	Subtraction Assignment	a-=b	a=6
=	Multiplication Assignment	a=b	a=112
/=	Division Assignment	a/=b	a=1
%=	Modulus Assignment	a%=b	a=6
++	Increment	c=a++	c=15
--	Decrement	c=b--	c=7

දැන් අපි ඉහත වගුව පිළිබඳ වැඩිදුරටත් අධ්‍යයනයක යෙදෙමු.

මුල් ඔපරේටර් 4 සම්බන්ධව පැහැදිලි කිරීමට අවශ්‍ය තැහැ ඒවා බැලූ පමණින් සිදුව ඇතිදෙය තේරුම් ගන්න පුළුවන්. අපි modulus ඔපරේටර් එකේ ඉඳලා බලමු.

modulus operator එකෙන් සිදුකරන්නේ යම් සංඛ්‍යාවක් තවත් සංඛ්‍යාවකින් බෙදුවීම ඉතිරිවන අගය ලබාදීමයි. උදාහරණයක් ලෙස a%b=6 ලෙස ඉහත වගුවේ දැක්වෙනවා, 14 සංඛ්‍යාව 8 න් බෙදුවීම 6 ක් ඉතිරි වනවා.

Addition Assignment operator එකෙන් a+=b ලෙස ලියන දෙයම a+=b ලෙස කෙටිකර ලියන්න පුළුවන්. අනිත් assignment operators සියල්ල මගින් සිදුවන්නේ එම ආකාරයේ ක්‍රියාවලියක්.

උදා :

a*=b -> a=a*b

a%=b -> a=a%b

පහත ජාවා වැඩසටහන තුළින් මූලික arithmetic operators කිහිපය ක්‍රියාකරන අයුරු තවත් පැහැදිලි කරගන්න පුළුවන් වෙයි.

1. class Operators{
2. public static void main(String args[]){
3. int a=14;
4. int b=8;
5. int tmp=0;
6. System.out.println("Value of a = " + a);
7. System.out.println("Value of b = " + b);
8. System.out.println("a + b = " + String.valueOf(a+b));
9. System.out.println("a - b = " + String.valueOf(a-b));
10. System.out.println("a * b = " + String.valueOf(a*b));
11. System.out.println("a / b = " + String.valueOf(a/b));
12. System.out.println("a % b = " + String.valueOf(a%b));
13. System.out.println("a ++ = " + String.valueOf(++a));
14. System.out.println("Value of a = " + a);
15. System.out.println("b -- = " + String.valueOf(b--));
16. System.out.println("Value of b = " + b);
17. }
18. }

පැහැදිලි කිරීම:

String.valueOf() method එක භාවිතා කරනුයේ integer ආකාරයෙන් ඇති ගණිතමය ප්‍රතිඵලය console එකට යැවීමට පෙර String ආකාරයට පත් කිරීමටයි.

ප්‍රතිඵලය:

```

C:\ Command Prompt

D:\Java Examples>java Operators
Value of a = 14
Value of b = 8
a + b      = 22
a - b      = 6
a * b      = 112
a / b      = 1
a % b      = 6
a ++       = 14
Value of a = 15
b --       = 8
Value of b = 7

D:\Java Examples>_

```

ජාවා ක්‍රමලේඛනයේ යෙදීමේදී උපයෝගී කරගත හැකි මෘදුකාංග

Saturday, April 10, 2010 by Kaniskha Dilshan

ජාවා වැඩසටහන් ලිවීමට මූලික වශයෙන් අත්‍යාවශ්‍ය වන්නේ command prompt එක(console), සරල text editor එකක් සහ JDK (Java Development KIT) එක පමණයි. සරල text editor එකක් ලෙස වින්ඩෝස් වලදී Notepad ද Linux වලදී VI Editor ද යොදාගත හැක. එහෙත් ආධුනිකයෙකුට මෙය ප්‍රමාණවත් වුවත් වැඩිදුරටත් ජාවා හැඳූරීමේ දී මීට වඩා පහසුකම් අවශ්‍යවේ. උදාහරණයක් වශයෙන් පේලි අංකය(line number) පෙන්වීම ක්‍රමලේඛනයේ යෙදීමේදී වැදගත් වේ. line indentation තැබිය හැකිවීම තවත් අවශ්‍යතාවයකි. syntax highlighting නිබේනම් තවත් පහසුකම්. මේවා සියල්ල සරල text editor එකකින් ලබා ගැනීම අසීරු වේ.

IDE (Integrated Development Environment) එකක් යොදා ගැනීම ඉතා විශාල පහසුවක් වුවත් ඒවා මගින් ක්‍රමලේඛකයා විසින් කළයුතු ඉතා විශාල කාර්යභාරයක් සිදුකරන නිසා ජාවා මූලික සංකල්ප ඉගෙන ගැනීමේ දී මේවා යොදා ගැනීම යෝග්‍ය නොවේ. Netbeans හා Eclipse යනු ඉතා ජනප්‍රිය හා නොමිලේ ලබාදෙන IDE වේ.

අපි දැන් බලමු ජාවා වැඩසටහන් ලිවීමට යොදාගත හැකි editors කිහිපයක්.



Notepad++

භාගත කරගැනීමට : <http://sourceforge.net/projects/notepad-plus/files/>
ක්‍රියාකාරී අවටාවක්:

```

D:\Java\Java Programming Assignment\Task 2\B\Complete Source Code\DBHandler.java - Notepad++
File Edit Search View Format Language Settings Macro Run Plugins Window Help
color codes.html Edit Java OpenSource.java SQL SQL report.java TestSimple.java DBHandler.java Edit

37 public ResultSet getTheResult(String SQLCommand)
38 {
39     try
40     {
41         //return con.createStatement().executeQuery(SQLCommand);
42         Statement stat=con.createStatement(ResultSet.TYPE_SCROLL_S
43         ResultSet rs1=stat.executeQuery(SQLCommand);
44         return rs1;
45     }
46     catch(SQLException ex)
47     {
48         JOptionPane.showMessageDialog(null,ex.getMessage(),"Error
49         return null;
50     }
51 }
52
53 //Method for execute a SQL Query (Updating deleting....)
54 public void executeSQL(String SQLCommand)
55 {
56     try
57     {
58         con.createStatement().executeQuery(SQLCommand);
59     }
60     catch(SQLException ex)
61     {

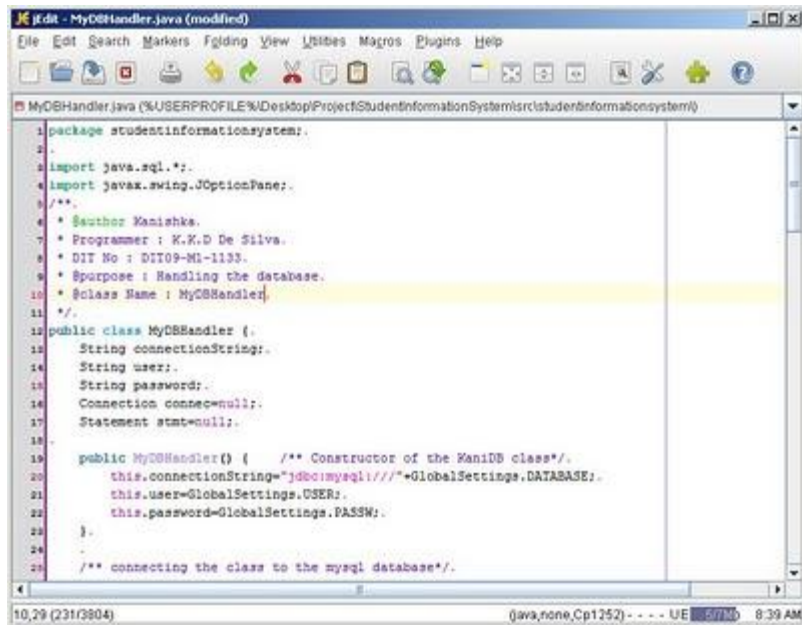
```



JEdit

භාගත කරගැනීමට : <http://www.jedit.org/>

ක්‍රියාකාරී අවටාවක්:



```
1 package studentinformationssystem;
2
3 import java.sql.*;
4 import javax.swing.JOptionPane;
5 /**
6  * @author Manishka.
7  * Programmer : K.K.D De Silva.
8  * DIT No : DIT09-M1-1133.
9  * @purpose : Handling the database.
10  * @class Name : MyDBHandler
11  */
12 public class MyDBHandler {
13     String connectionString;
14     String user;
15     String password;
16     Connection conn=null;
17     Statement stmt=null;
18
19     public MyDBHandler() { /** Constructor of the ManiDB class*/
20         this.connectionString="jdbc:mysql://"+GlobalSettings.DATABASE;
21         this.user=GlobalSettings.USER;
22         this.password=GlobalSettings.PASSW;
23     }
24
25     /** connecting the class to the mysql database*/
```

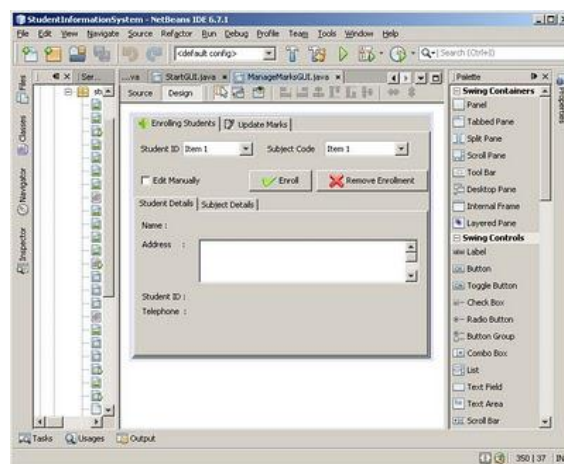
නොමිලයේ ලබාදෙන IDE



Netbeans

භාගත කරගැනීමට : <http://netbeans.org/>

ක්‍රියාකාරී අවටාවක්:

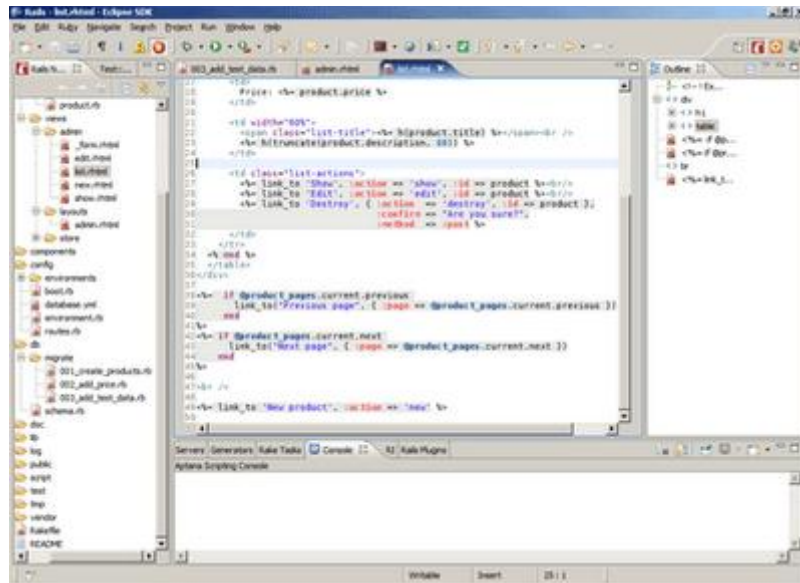




Eclipse

භාගත කරගැනීමට : <http://www.eclipse.org/downloads/>

ක්‍රියාකාරී අවටාවක්:



ජාවා මූලික සංකල්ප ගැන හොඳ අවබෝධයක් ලැබුන විට Netbeans , Eclipse වැනි IDE එකක් භාවිතා කිරීම සුදුසුය එතෙක් .JEdit, Notepad++ වැනි editor එකක් භාවිතා කිරීම නුවණට හුරුය.

ජාවා කාරක (Java Operators) 2 කොටස :: Relational Operators

by Kaniskha Dilshan

මීට පෙර ජාවා ඔපරේටර් ගැන කල පාඩමෙන් සාකච්ඡා වුනේ arithmetic operators පිළිබඳවයි. මම මෙම පාඩමෙන් පහදා දීමට උත්සාහ කරන්නේ relational operators පිළිබඳවයි. relational operators යොදා ගැනෙන්නේ කිසියම් අගයන් හෝ විචල්‍යයන් අතර සම්බන්ධය ඇගයීමටයි(evaluate).

උදාහරණයක් වශයෙන් a හා b ලෙස විචල්‍යයන් 2 ක් ඇතුළු සිතන්න. a හා b සමානද, a හා b අසමානද, a, b ට වඩා විශාලද, a, b ට වඩා කුඩාද, a, b ට වඩා විශාල හෝ සමානද, a, b ට වඩා කුඩා හෝ සමානද යන්න පරීක්ෂා කිරීම සඳහා අපට relational operators යොදාගත හැකිය. relational operators සෑම විටම ප්‍රතිඵලය ලබා දෙනුයේ boolean ආකාරයෙනි.

අපි පහත වගුව තුළින් relational operators පිළිබඳ තවදුරටත් බලමු.

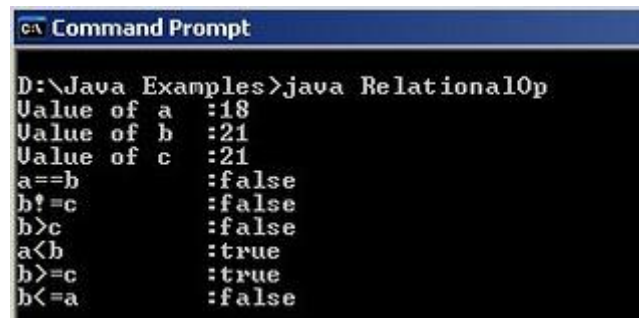
මෙහිදී මම a=18,b=21,c=21 ලෙස අගයන් ආදේශ කරඇති integer විචල්‍යයන් 3 ක් ගෙන ඇතිබව සලකන්න.

(operator)	operator name	usage	result
==	Equal to	a==b	false
!=	Not equal to	b!=c	false
>	Greater than	b>c	false
<	Less than	a<b	true
>=	Greater than or equal to	b>=c	true
<=	Less than or equal to	b<=a	false

මෙය තවත් තහවුරු කරගැනීම සඳහා ලියන ලද සරල ජාවා වැඩසටහන පහත පරිදිවේ.

```
1. class RelationalOp{
2.     public static void main(String args[]){
3.         int a=18;
4.         int b=21;
5.         int c=21;
6.         System.out.println("Value of a      : " + (a));
7.         System.out.println("Value of b      : " + (b));
8.         System.out.println("Value of c      : " + (c));
9.         System.out.println("a == b         : " + (a==b));
10.        System.out.println("b != c          : " + (b != c));
11.        System.out.println("b > c           : " + (b > c));
12.        System.out.println("a < b           : " + (a < b));
13.        System.out.println("b >= c          : " + (b >= c));
14.        System.out.println("b <= a          : " + (a == b));
15.    }
16. }
```

ප්‍රතිඵලය:



ඔබට පහත ජාවා applet එක භාවිතයෙන් මෙම relational operators භාවිතා වන ආකෘය තවත් පැහැදිලි වනු ඇති.

අපි ඊළඟ පාඩමෙන් බලමු Boolean logical operators ක්‍රියා කරන ආකාරය.

ජාවා කාරක (Java Operators) 3 කොටස :: Boolean Logical Operators

Sunday, April 11, 2010 by Kaniskha Dilshan

අපි මීට පෙර පාඩමෙන් කතා කළේ relational operators ගැනයි. මෙම පාඩමෙන් boolean logical operators ගැන ඉගෙන ගනිමු. තාර්කික අවශ්‍යතා ඉටුකර ගැනීම සඳහා මෙම ඔපරේටර් වර්ගය යොදා ගත හැකිය. අපි පහත වගු තුළින් මෙම ඔපරේටර් පිළිබඳව අධ්‍යයනය කරමු.

Boolean Logical Operators

Operator	Name
&	Logical AND
	Logical OR
^	Logical XOR
&&	Short Circuit AND
	Short Circuit OR
!	Logical NOT
&=	AND Assignment
=	OR Assignment
^=	XOR Assignment
==	Equal To
!=	Not Equal To
? :	Ternary operator (if - then - else)

&& (AND Operator)

පහත වගුව තුළින් මෙහි ක්‍රියාකාරීත්වය අධ්‍යයනය කරමු

syntax : **A && B**

A	B	A && B
false	false	false
false	true	false
true	false	false
true	true	true

|| (OR Operator)

පහත වගුව තුළින් මෙහි ක්‍රියාකාරීත්වය අධ්‍යයනය කරමු

syntax : **A || B**

A	B	A B
false	false	false
false	true	true
true	false	true
true	true	true

^ (XOR Operator)

පහත වගුව තුළින් මෙහි ක්‍රියාකාරීත්වය අධ්‍යයනය කරමු

syntax : **A ^ B**

A	B	A ^ B
false	false	false
false	true	true
true	false	true
true	true	false

! (NOT Operator)

A	!A
false	true
true	false

දැන් බලමු Logical AND සහ Short Circuit AND ඔපරේටර් අතර වෙනස කුමක්ද කියලා.

Logical AND ඔපරේටර් එකේදී ඔපරේටර් එකට දෙපසම ඇති conditions අනිවාර්යයෙන්ම ඇගයීම (evaluate) සිදුවනවා. නමුත් Short Circuit AND ඔපරේටර් එකේදී වම් පස conditions එක false නම් දකුණුපස condition එක ඇගයීමකින් තොරවම අවසන් ප්‍රතිඵලය false ලෙස සටහන් කරනවා, මන්දයත් AND මගින් false return කිරීමට එක් condition එකක් false වීම ප්‍රමාණවත් බැවිණි.

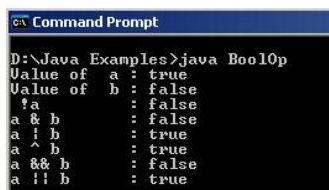
මෙලෙසින්ම Short Circuit OR ඔපරේටර් එකේදී වම් පස condition එක true ලෙස පවතිනම් දකුණුපස condition එක ඇගයීමකින් තොරවම අවසන් ප්‍රතිඵලය true ලෙස සටහන් කරනවා. මන්දයත් OR මගින් true return කිරීමට එක් condition එකක් true වීම ප්‍රමාණවත් බැවිණි.

මෙලෙස Short Circuit ඔපරේටර් භාවිතා කිරීම මගින් වියාල වශයෙන් ජාවා වැඩසටහනක කායික්ශමතාව වැඩි කල හැකිය. මේ නිසා මේ ඔපරේටර් දෙක අතුරින් ජාවාහි conditional operator එකක් ලෙස බහුලව භාවිතාවනුයේ Short Circuit ඔපරේටර්ය.

Boolean Logical Operators හි ක්‍රියාකාරීත්වය අධ්‍යයනය කිරීමට ලියන ලද ජාවා වැඩසටහනක් පහත දැක්වේ.

```
1. class BoolOp {
2.     public static void main(String args[]){
3.         boolean a=true;
4.         boolean b=false;
5.         System.out.println("Value of a : " +a);
6.         System.out.println("Value of b : " +b);
7.         System.out.println(" !a      : " + (!a));
8.         System.out.println("a & b    : " + (a & b));
9.         System.out.println("a | b    : " + (a | b));
10.        System.out.println("a ^ b    : " + (a ^ b));
11.        System.out.println("a && b   : " + (a && b));
12.        System.out.println("a || b   : " + (a || b));
13.    }
14. }
```

ප්‍රතිඵලය :



```

D:\Java Examples>java BoolOp
Value of a : true
Value of b : false
!a          : false
a & b       : false
a | b       : true
a ^ b       : true
a && b      : false
a || b      : true
```

ඡායා Selection Statements

Saturday, May 1, 2010 by Kaniskha Dilshan

ඕනෑම පරිගණක ක්‍රම ලේඛනයක් සාමාන්‍යයෙන් මූලික කොටස් 3 කින් සමන්විත වෙනවා

1. Selection
2. Sequence
3. Iteration

අප මෙම පාඩමේදී සලකා බලන්නේ ඡායාහි selection භාවිතා වන්නේ කුමන ආකාරයෙන්ද යන්නයි. ඡායා තුල selection සඳහා සහය දැක්වීමට පහත සඳහන් විකල්ප 6 භාවිතා කරන්න පුළුවන්.

- if
- if - else
- Nested if
- if - else - if
- switch statement
- Ternary operator

if statement

syntax :

1. if (condition) {
2. //Perform Task1
3. }

මෙහිදී සිදුවන්නේ **condition** යන්න true ලෙස පවතීනම් if code block එක තුල ඇති statements (Task1) execute කිරීමයි.

if - else statement

syntax :

1. if(condition){
2. //Perform Task1
3. }else{
4. //Perform Task2
5. }

මෙහිදී සිදුවන්නේ **condition** යන්න true ලෙස පවතීනම් if code block එක තුල ඇති statements (Task1) execute කිරීමයි. **condition** යන්න false ලෙස පවතීනම් else සඳහා අදාල code block එක තුල ඇති statements (Task2) execute කිරීමයි.

Nested if statements

syntax :

1. if(condition1){
2. if(condition2){
3. //Peform Task1
4. }else{
5. //Peform Task2
6. }
7. }else{
8. //Peform Task3
9. }

මෙහිදී සිදුකර ඇත්තේ if statement කිහිපයක් එකට ගොනු කර දැක්වීමයි. condition1 තෘප්ත වේ නම් එම if block එක තුල ඇති ඊළඟ if statement එක execute වේ එනම් condition2 තෘප්ත වෙනම් Task1 සිදුකරයි. condition2 යන්න තෘප්ත නොවේ නම් Task2 සිදුකරයි. යම් හෙයකින් condition1 තෘප්ත නොවේ නම්(false) Task3 සිදුකරයි. දැන් ඔබට පැහැදිලි වනවා ඇති Task1 සිදුවීමට නම් condition1 සහ condition2 යන දෙකම තෘප්ත විය යුතු බව. Task2 සිදුවීමට condition1 තෘප්ත වීම ප්‍රමාණවත්ය. condition1 තෘප්ත නොවීම සිදුවුවහොත් Task3 අනිවාර්යයෙන්ම සිදුවනු ඇත.

if - else - if statements

මෙහිදී අපට මුලින් ඇති if condition එක තෘප්ත නොවේ නම් තවත් if conditions කිහිපයක් යෙදිය හැක.

syntax:

1. if(condition1){
2. //Perform Task1

```

3.     }else if(condition2){
4.         //Perform Task2
5.     }else if(condition3){
6.         //Perform Task3
7.     }else{
8.         //Perform Task4
9.     }

```

switch statement

ජාවා තුළදී switch statement එක සහය දක්වන්නේ int,byte,short,char සහ enumeration data types සඳහා පමණි.

ex:

```

1.     class Case {
2.     public static void main(String args[]){
3.
4.         int week=3;
5.         String output;
6.         switch(week){
7.             case 1 : output="Sunday";break;
8.             case 2 : output="Monday";break;
9.             case 3 : output="Tuesday";break;
10.            case 4 : output="Wednesday";break;
11.            case 5 : output="Thursday";break;
12.            case 6 : output="Friday";break;
13.            case 7 : output="Saturday";break;
14.            default : output="Invalid day of week!";
15.        }//end of the switch statement
16.
17.        System.out.println(output);
18.    }
19. }

```

මෙහිදී break keyword එක යොදා තිබෙන්නේ condition එක තෘප්ත වූ විට switch block එකෙන් ඉවතට යාමටයි. default යටතේ දී ඇති statement එක execute වනුයේ ඉහත සඳහන් කිසිදු case එකක් සමග දී ඇති values නොගැලපුනහොත් පමණි.

Ternary operator

syntax:

```

1.     result=Condition ? valueA : valueB

```

මෙහිදී සිදුවන්නේ දී ඇති condition එක true නම් result සඳහා valueA ද නො එසේ නම් valueB ද assign කිරීමයි.

example :

```

1.     int max;
2.     int a=12 , b=54;
3.     max = (a>b) ? a : b ;

```

ජාවා Iterations (loops)

Sunday, May 2, 2010 by Kaniskha Dilshan

අපි මේ පාඩමෙන් සාකච්ඡා කරන්න යන්නේ ජාවාහි ඉතාම වැදගත් කොටසක් ගැන. එනම් Iterations (නැවත නැවත සිදුකිරීම්) ජාවා වලදී පමණක් නොවෙයි ඕනෑම ක්‍රමලේඛන භාෂාවකදී Iterations හෙවත් loops ඉතා වැදගත් වනවා. Iterations භාවිතා වන අවස්ථා වලට උදාහරණ:

- කිසියම් array එකක් තුළ ඇති elements (අවයව) එකින් එක පිළිවෙලින් පරීක්ෂා කිරීම.
- විවිධ ඇල්ගොරිතම (algorithm) නිර්මාණයේදී.
- කිසියම් දත්ත සමූහයක් විශ්ලේෂණය කිරීමේදී

අපි දැන් බලමු ජාවාතුල භාවිතවන looping structures මොනවාද කියලා.

- while
- do - while
- for
- for - each

while Loop

syntax:

```
1. while(expression){
2.    //do something
3. }
```

මෙහිදී expression එක true ලෙස පවතින තාක් loop එක ක්‍රියාත්මක වීම සිදුවේ (නැවත නැවත සිදුවේ). අපි දැන් while loop එක ක්‍රියාත්මක වන අන්දම විස්තරාත්මකව බලමු.

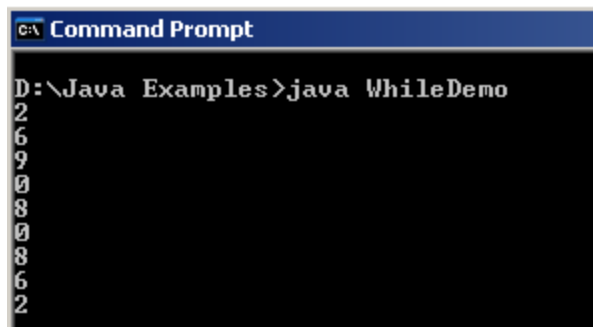
මෙහිදී expression එකෙහි boolean අගය පරීක්ෂා කිරීම සිදුවනවා. එය false නම් while එක ඇතුළේ ඇති statement එක execute කිරීම සිදුනොකර කෙලින්ම loop එකෙන් පිටතට යාම සිදුකරනවා. එය true නම් while loop එක ඇතුළේ ඇති statement execute කර අවසානයේ නැවත expression එක පරීක්ෂා කරනවා. මේ ආකාරයට දිගින් දිගටම මෙය සිදුවනවා. සාමාන්‍යයෙන් while loop එකක් යොදාගන්නේ loop එක කොපමණ වාර ගණනක් ක්‍රියාත්මක විය යුතුද යන්න කලින් අනුමාන කල නොහැකි අවස්ථා වලදීය.

අපි සරල ජාවා වැඩසටහනකින් බලමු while iteration එකක් ක්‍රියා කරන ආකාරය.

ex :

```
1. class WhileDemo {
2.     public static void main(String args[]){
3.         int val=(int)(Math.random()*10);
4.         while(val !=5){
5.             System.out.println(val);
6.             val=(int)(Math.random()*10);
7.         }
8.     }
9. }
```

output :



මෙහිදී සසම්භාවී ලෙස 0 සිට 9 තෙක් පූර්ණ සංඛ්‍යා උත්පාදනය කොට ලැබුනු සංඛ්‍යාව 5 නම් loop එක නැවැත්වීම සිදුකරනවා. මෙහිදී අපිට 5 කිවෙති සංඛ්‍යාව ලෙස ලැබෙද්දී කිව නොහැකියි. එමනිසා මෙවැනි අවස්ථාවකදී while loop එක උපයෝගී කරගැනීම පහසුයි. ඉහත උදාහරණයේ 2,6,9,0,8,0,8,6,2 යන සංඛ්‍යා පිළිවෙලින් ලැබී අවසානයේ ලැබී ඇත්තේ 5 යි. එමනිසා while loop එක නැවතීම සිදුවී තිබෙනවා.

do - while loop

syntax:

```
1. do{
2.    //do some thing
3. }while(expression);
```

මෙහිදී expression එක පරීක්ෂා කිරීම සිදුකරන්නේ statements execute කිරීමෙන් පසුවයි.

example :

```
1. class DoWhileDemo {
2.     public static void main(String args[]){
3.         int val=(int)(Math.random()*10);
4.         do{
5.             System.out.println(val);
6.             val=(int)(Math.random()*10);
```

```

7.     }while(val !=5);
8.     }
9.     }

```

output :

```

C:\ Java Console
D:\Java Examples>java DowhileDemo
5
7
4
2

```

මෙහිදී generate කරන ලද සංඛ්‍යාව 5 ට අසමානද යන්න පරීක්ෂා කරනුයේ loop එක අවසානයේ දීය. console එකෙහි මුලින් 5 print වීමට හේතුව මෙයයි.

for loop
syntax:

```

1.     for(initialization ; Expression ; UpdateStatement ){
2.         //do something
3.     }

```

example :

```

1.     int tot=0;
2.     for(int i=1;i<=30;i++){
3.         tot+=i;
4.     }

```

සාමාන්‍යයෙන් for loop එකක් භාවිතා කරන්නේ loop එක ක්‍රියාත්මක වන වාර ගණන දන්නා විටදීය.

for - each loop
syntax :

```

1.     for(element : array ){
2.         //do something
3.     }

```

මෙහිදී දෙනලද array එකක් තුළ ඇති සියලු අවයව (elements) එකින් එක ලබාගැනීම කළ හැකියි. දී ඇති element නම් variable එක තුළට සෑම iteration එකක දීම value ලබාදීම සිදුකරනවා. අපි සරළ ජාවා වැඩසටහනක් තුලින් බලමු for - each iteration එක ක්‍රියා කරන ආකාරය.

example :

```

1.     class ForEachDemo{
2.     public static void main(String args[]){
3.         int arr[]={33,45,67,78,56};
4.         int tot=0;
5.         for(int i:arr){
6.             System.out.println(i);
7.             tot=tot+i;
8.         }
9.         System.out.println("Sum : " + tot );
10.    }
11.    }

```

output :

```

C:\ Command Prompt
D:\Java Examples>java ForEachDemo
33
45
67
78
56
Sum : 279

```

ජාවා Jump Statements සහ loops සඳහා අභ්‍යාස.

Monday, May 3, 2010 by Kaniskha Dilshan

ජාවා සඳහා jump statement 3 ක් සහය දක්වනවා. මේවා වැදගත් වනුයේ අපට කිසියම් control structure එකක් පාලනය කිරීමට අවශ්‍ය විටදීයි. උදාහරණයක් ලෙස සැලකුවහොත් අපට කිසියම් loop එකක මැදදී එය නැවැත්වීමට අවශ්‍ය වුවහොත් අපට මෙම jumping statements යොදාගන්න පුළුවන්.

- break
- continue
- return

break

example :

```
1. class BreakDemo {
2.     public static void main(String args[]){
3.         for(int i=0;i<10;i++){
4.             if(i==5){
5.                 break; //if i is equal to 5 terminate the loop
6.             }
7.             System.out.print(i + " ");
8.         }
9.     }
10. }
```

output :



මෙහිදී සිදුවන්නේ ක්‍රමයෙන් වැඩිවිගෙන යන i අගය 5 ට සමාන වූ විගස loop එක නැවැත්වීමයි. break keyword එකෙන් විධාන කරනුයේ loop එකෙන් පිටතට යෑමටයි. එනම් loop එක නැවැත්වීමයි.

continue

```
1. class ContinueDemo {
2.     public static void main(String args[]){
3.         for(int i=0;i<10;i++){
4.             if(i==5){
5.                 continue; //if i is equal to 5 ignore i and continue
6.             }
7.             System.out.print(i + " ");
8.         }
9.     }
10. }
```

output :



මෙහිදී සිදුවන්නේ ක්‍රමයෙන් වැඩිවිගෙන යන i අගය 5 ට සමාන වූ විට continue keyword එකට යටින් ඇති කිසිදු statement එකක් execute නොකර loop එකෙහි ඊළඟ step එකට පැනීමයි. එනම් i හි අගය 6 ට අදාළ iteration එක පටන් ගැනීමයි. මෙහිදී 5 යන අගය console එකෙහි මුද්‍රණය නොවීමට හේතුව දැන් ඔබට වටහා ගත හැකිවිය යුතුය.

return

මෙම keyword එක පිළිබඳ අපි method සම්බන්ධ පාඩමේදී සවිස්තරාත්මකව සාකච්ඡා කරමු.

loops සඳහා අභ්‍යාස

පහත රටාවන් ලබාගැනීමට loops යොදාගන්න.

```
C:\ Command Prompt
D:\Java Examples\Exercise 01>java Pattern01
#####
#####
#####
#####
#####
#####
#####
#####
#####
#####
```

```
1. class Pattern01{
2.     public static void main(String args[]){
3.         for(int i=0;i<10;i++){
4.             for(int j=0;j<10;j++){
5.                 System.out.print("#");
6.             }
7.             System.out.println();
8.         }
9.     }
10. }
```

```
C:\ Command Prompt
D:\Java Examples\Exercise 01>java Pattern03
#####
#####
#####
#####
#####
#####
#####
#####
#####
#####
```

```
1. class Pattern03{
2.     public static void main(String args[]){
3.         for(int i=0;i<10;i++){
4.             for(int j=10-i;j>0;j--){
5.                 System.out.print("#");
6.             }
7.             System.out.println();
8.         }
9.     }
10. }
```

```
C:\ Command Prompt
D:\Java Examples\Exercise 01>java Pattern04
      ##
     ##
    ##
   ##
  ##
 ##
##
##
##
##
##
##
##
##
```

```
1. class Pattern04{
2.     public static void main(String args[]){
3.         for(int i=0;i<10;i++){
4.             for(int j=10-i;j>0;j--){
5.                 System.out.print(" ");
6.             }
7.             for(int j=0;j<=i;j++){
8.                 System.out.print("#");
9.             }
10.         }
11.     }
12. }
```



```

10.     System.out.println();
11.     }
12.     }
13.     }

```

```

C:\> Command Prompt

D:\Java Examples\Exercise 01>java Pattern05

  #
 ###
#####
#####
#####
#####
#####
#####
#####
#####
#####
#####

```

```

1.     class Pattern05{
2.     public static void main(String args[]){
3.     for(int i=0;i<10;i++){
4.     for(int j=10-i;j>0;j--){
5.     System.out.print(" ");
6.     }
7.     for(int j=1;j<=i*2-1;j++){
8.     System.out.print("#");
9.     }
10.    System.out.println();
11.    }
12.    }
13.    }

```

```

C:\> Command Prompt

D:\Java Examples\Exercise 01>java Pattern06

#####
#####
#####
#####
#####
#####
#####
#####
#####
#####

```

```

1.     class Pattern06{
2.     public static void main(String args[]){
3.     for(int i=0;i<10;i++){
4.     for(int j=10-i;j>0;j--){
5.     System.out.print(" ");
6.     }
7.     for(int j=0;j<10;j++){
8.     System.out.print("#");
9.     }
10.    System.out.println();
11.    }
12.    }
13.    }

```

```

C:\> Command Prompt

D:\Java Examples\Exercise 01>java Pattern07

 1
123
12345
1234567

```

```

1. class Pattern05{
2.     public static void main(String args[]){
3.         for(int i=0;i<5;i++){
4.             for(int j=5-i;j>0;j--){
5.                 System.out.print(" ");
6.             }
7.             for(int j=1;j<=i*2-1;j++){
8.                 System.out.print(j);
9.             }
10.            System.out.println();
11.        }
12.    }
13. }

```

```

D:\Java Examples\Exercise 01>java Pattern05
1      2      3      4      5      6      7      8      9
2      4      6      8      10     12     14     16     18
3      6      9      12     15     18     21     24     27
4      8      12     16     20     24     28     32     36
5      10     15     20     25     30     35     40     45
6      12     18     24     30     36     42     48     54
7      14     21     28     35     42     49     56     63
8      16     24     32     40     48     56     64     72
9      18     27     36     45     54     63     72     81

```

```

1. class Pattern08{
2.     public static void main(String args[]){
3.         for(int i=1;i<10;i++){
4.             for(int j=1;j<10;j++){
5.                 System.out.print(i*j + "\t" );
6.             }
7.             System.out.println();
8.         }
9.     }
10. }

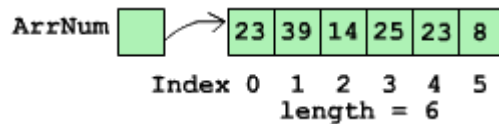
```

ජාවා තුල Arrays භාවිතය I

Tuesday, May 4, 2010 by Kaniskha Dilshan

Array එකක් යනු සරලම ආකාරයේ දත්ත ව්‍යුහයක් (Data Structure) ලෙස දක්වන්න පුලුවන්. අපි මීට පෙර විචල්‍යයන්(variables) ගැන කතා කලා. Array එකක් යනු එකම දත්ත ආකාරයක්(data type) ඇති විචල්‍යයන් සමූහයක්. මෙහිදී array එකකට නිශ්චිත විශාලත්වක් ඇතිබව විශේෂයෙන් සඳහන් කල යුතුයි.

Array එක සමන්විත වන variables, elements යනුවෙන් හඳුන්වනවා. Array එක තුල element එකක් හඳුනා ගැනීමට භාවිතා වන්නේ index එකයි. index එකෙහි ආරම්භක අගය 0 වනවා. අපි පහත සටහන ආධාරයෙන් array එකක ව්‍යුහය පිළිබඳ අධ්‍යයනය කරමු.



ArrayNum යනු array එක සඳහා දී ඇති නමයි. 0 වන index එක ලෙස 23 ද, අවසාන index එක (5) ලෙස 8 ද මම මෙහිදී ආදේශ කර තිබෙන්නවා. Array එකේ size එක (length) ලෙස 6 පවතිනවා(මුළු elements ගණන).

ArrNum යන්න RAM එකෙහි ස්ථාපනය වී ඇති array එකට pointer එකක් හා සමානයි. නමුත් java pointer arithmetic සඳහා සහාය නොදක්වන බව මතක තබාගත යුතුයි.

අපි දැන් බලමු ජාවා වල array එකක් create කරගත හැකි ආකාර.

syntax for creating arrays:

method 01

1. `DataType ArrayName[];`
2. `ArrayName=new DataType[SIZE];`

Ex :

1. `int MyArray[];`
2. `MyArray=new int[5];`

method 02

1. `DataType ArrayName[]=new DataType[SIZE];`

Ex :

1. `int MyArray=new int[5];`

method 03

1. `DataType ArrayName[]={ value1,value2,value3,value_n};`

Ex :

1. `int MyArray[]={ 23,45,67,23,-32,66};`

method 01 හා method 02 භාවිත කර arrays සෑදීමේදී array එකේ size එක(elements ගණන) ලබා දිය යුතු වුවත් method03 භාවිත කිරීමේදී එසේ අවශ්‍ය නැ. Array එකක් සෑදූ පසු එහි size එක පසුව වෙනස් කිරීම කල නොහැකියි.

අපි දැන් බලමු සකසන ලද array එකකට values assign කරන ආකාරය.
example:

1. `MyArray[0]=44;`
2. `MyArray[4]=68;`

මෙහිදී අප සකසන ලද array එකේ උපරිම index එකට වඩා වැඩි අගයක් index ලෙස ලබා දුන්නහොත් [ArrayIndexOutOfBoundsException](#) එකක් ලබාදීම සිදුවනවා. අපි exception පිළිබඳ ඉදිරි පාඩමකදී පුළුල්ව සාකච්ඡා කරමු.

Java Arrays හා සම්බන්ද length property එක

ජාවාහි සෑම array එකකටම length නම් property එකක් තිබෙන්නවා. මෙමගින් අපිට array එකක size එක(number of elements) ලබාගන්න පුලුවන්. මෙම නිසා C/C++ වලදී array භාවිතාකිරීමට වඩා java තුලදී arrays භාවිතා කිරීම පහසු කරනවා.

උදා : C වලදී function එකකට array එකක් pass කරන විට එහි size එකත් වෙනම parameter එකක් මගින් pass කළ යුතුවනවා. නමුත් ජාවා තුලදී length property එක මගින් size එක ලබාගත හැකිනිසා එසේ size එක වෙනම variable එකකින් pass කළ යුතු නැහැ. array එක තුළින්ම එහි size එක ලබාගන්න පුළුවන්.

අපි දැන් ඉහත සටහනේ ඉදිරිපත් කරඇති array එක ජවා තුල නිර්මාණය කරගැනීම සඳහා සුදුසු කේතයක් ලියමු.
Code 01

```
1.    int ArrNum[]={23,39,14,25,23,8};
```

Code 02

```
1.    int ArrNum[]=new int[6];
2.    ArrNum[0]=23;
3.    ArrNum[1]=39;
4.    ArrNum[2]=14;
5.    ArrNum[3]=25;
6.    ArrNum[4]=23;
7.    ArrNum[5]=8;
```

දැන් අපි බලමු ඉහත සකස් කරගත් Array එකෙහි values පරිශීලනය(access) කරන ආකාරය.
Access values one by one:

```
1.    int value_at_index_4 = ArrNum[4];
2.    int value_at_index_1 = ArrNum[1];
```

Access values using a for loop:

```
1.    for(int i=0;i < ArrNum.length ; i++){
2.        int value_at_index_i=ArrNum[i];
3.        System.out.println(value_at_index_i);
4.    }
```

Access values using a for-each loop:

```
1.    for(int elem:ArrNum){
2.        int value_at_index_i=elem;
3.        System.out.println(value_at_index_i);
4.    }
```

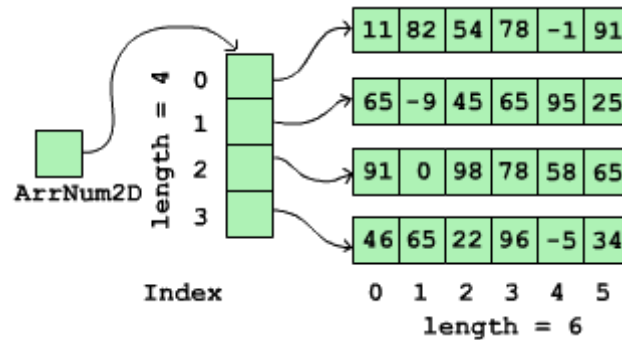
Access values using a while loop:

```
1.    int i=0;
2.    while( i < ArrNum.length ){
3.        int value_at_index_i=ArrNum[i];
4.        System.out.println(value_at_index_i);
5.        i++;
6.    }
```

ජාවා තුල Arrays භාවිතය II (2D Arrays)

Wednesday, May 12, 2010 by Kaniskha Dilshan

බොහෝ විට 1-dimensional arrays භාවිතා වුවද 2-dimensional arrays ද වැදගත්වන අවස්ථා බොහොමයකි. උදා matrix සම්බන්ධ ගණනය කිරීම් වලදී. table එකක් වැනි යම් නිරූපණය කිරීමටද 2D arrays භාවිතා කළ හැක. අපි මීට පෙර පාඩමෙන් arrays පිළිබඳ මූලික සංකල්ප 2D arrays වෙතද වලංගුවේ. අපි පහත රූප සටහන තුළින් බලමු ජාවා 2D array එකක් පරිගණකයේ ප්‍රධාන මතකය තුල තැන්පත් කරවන ආකාරය.



ජාවා තුල 2D arrays නිර්මාණය කිරීමේ ආකාර කිහිපයක් පවතී.

Syntax 01:

1. `DataType arrayName[][]=new DataType[ROWS][COLUMNS];`

ex :

1. `Float weight[][]=new float[5][3];`

Syntax 02:

1. `DataType arrayName[][]={ {row_1_val_1,row_1_val_2,...,row_1_val_n},`
2. `{row_2_val_1,row_2_val_2,...,row_2_val_n},`
3. `{row_3_val_1,row_3_val_2,...,row_3_val_n},`
4. `{row_n_val_1,row_n_val_2,...,row_n_val_n}};`

අපි දැන් බලමු ඉහත array එක ජාවා තුළින් නිර්මාණය කිරීමට සරළ වැඩසටහනක්.

```
1. class Arrays2D {
2.     public static void main(String args[]){
3.         int array2D[][]={ {11,82,54,78,-1,91} ,
4.                             {65,-9,45,65,95,25} ,
5.                             {91,0,98,78,58,65} ,
6.                             {46,65,22,96,-5,34} };
7.         //accessing array elements
8.         for(int row=0;row < array2D.length;row++){
9.             for(int col=0;col < array2D[row].length;col++){
10.                //System.out.print("Row : " + row + " Column : " + col );
11.                //System.out.println(" Value : " + array2D[row][col]);
12.                System.out.print(array2D[row][col]+ " ");
13.            }
14.            System.out.println();
15.        }
16.    }
17. }
```

output :

```

C:\Java Examples>java Arrays2D
11      82      54      78      -1      91
65      -9      45      65      95      25
91      0       98      78      58      65
46      65      22      96      -5      34
```

මේ සංකල්පයම භාවිතාකරමින් 3-dimensional arrays එකක් වූනත් හදාගන්න පුලුවන්. පහත ඉදිරිපත් කර ඇති නිදසුන බලන්න.

```

1.  class Array3D {
2.      public static void main(String args[]){
3.          //defining a 3D array
4.          float arrXD[][][]=new float[4][7][8];
5.          for(int i=0;i < arrXD.length;i++){
6.              for(int j=0;j < arrXD[i].length;j++){
7.                  for(int k=0;k < arrXD[i][j].length;k++){
8.                      //filling the 3D with random values
9.                      arrXD[i][j][k]=(float)Math.random();
10.                     //printing stored values
11.                     System.out.println(arrXD[i][j][k]);
12.                 }
13.             }
14.         }
15.     }
16. }

```

හොඳයි, අපි මීලඟ පාඩමේ සිට ප්‍රධාන පරිගණක භාෂාවේ ඉතාම වැදගත් කොටසක් වන වස්තු පාදක ක්‍රම ලේඛනය (Object Oriented Programming) පිළිබඳව අධ්‍යයනය කරමු.

Java වස්තු පාදක ක්‍රමලේඛනය I

Friday, May 14, 2010 by Kaniskha Dilshan

පරිගණක ක්‍රමලේඛනයේදී භාවිතා වන මූලික ආකාර දෙකක් තමයි

1. Structural/Procedural
2. Object-Oriented

structural ආකාරය භාවිතාවන පරිගණක භාෂා වලට උදාහරණ වශයෙන් C,Pascal,FORTRAN,COBOL වැනි පරිගණක භාෂා ඉදිරිපත් කරන්න පුලුවන්. මෙම භාෂා යොදාගෙන යම් ගැටලුවක් විසඳීමේදී එම ගැටලුව විසඳීමට අදාල පියවර(steps) පිළිබඳ තදින් සැලකිලිමත් වියයුතුයි. මේ සඳහා බොහෝවිට ඇල්ගොරිතමයන් භාවිතාවෙනවා. මෙම පියවරයන්හිදී සුදුසු data structures භාවිතා කිරීමද සිදුවනවා.

object-oriented ආකාරය භාවිතාවන පරිගණක භාෂා වලට උදාහරණ වශයෙන් Java,C++,Smalltalk හඳුන්වන්න පුලුවන් මෙම භාෂාවන් යොදාගෙන ගැටලු විසඳීමේදී තදින් සැලකිලිමත් වනුයේ වස්තූන්(entities/objects) පිළිබඳවයි. එහිදී එක් එක් වස්තුවට තිබිය යුතු ක්‍රියාකාරීත්වය(functionality) සහ වස්තූන් අතර සම්බන්ධතාවය යන කරුණු මූලික වශයෙන්ම සැලකිල්ලට ගනු ලබනවා. මෙහිදී ගැටලුව විසඳීමේ පියවර තීරණය කරණු ලබන්නේ object නොහොත් entity එකේ සැලැස්මයි(design).

මෙම අදහස ලඝු කොට දක්වන්නේ නම්:> **structural** ආකාරයට පරිගණක වැඩසටහන් සැකසීමේදී **step** එක අනුව **data structures** සැලසුම් කෙරෙනවා. **object-oriented** ආකාරයට පරිගණක වැඩසටහන් සැකසීමේදී **object** එකට අනුව **step** (පියවර) තීරණය වෙනවා.

ඉහත මූලික ආකාර 2 ට අමතරව functional,visual,4GL සහ තවත් ආකාර කිහිපයක් භාවිතා වනවා. නමුත් අප සාකච්ඡා කරන්නේ object-oriented ආකාරය පිළිබඳවයි.

object-oriented (වස්තු පාදක) සංකල්පය තවදුරටත් පැහැදිලි කරගැනීම සඳහා අපි ප්‍රායෝගික උදාහරණයක් දෙසට හැරෙමු. අපි කවුරුත් දන්නා මෙහෙයුම් පද්ධති දෙකක් වන DOS සහ Windows පිළිබඳ සලකා බලමු. DOS මෙහෙයුම් පද්ධතිය භාවිතා කරණුයේ procedural/structured (පටිපාටිගත) ආකාරයක්. අපිට යම්තැනක ඇති file එකක් copy කිරීමට අවශ්‍ය වූයේ නම් අපි කරන්නේ copy srcfile destfile ආකාරයේ විධානයක් භාවිතාකිරීමයි. නමුත් windows මෙහෙයුම් පද්ධතියේදී අපි කරන්නේ file එක ඇති ස්ථානයට ගොස්, එය මත right click කොට ලැබෙන popup මෙනුවෙන් copy select කොට paste කිරීමට අවශ්‍ය ස්ථානය කරා එලඹ එය paste කරගැනීමයි. මෙය සැබෑ ලෝකයේදී file එකක් සමග ගනුදෙනු කරන ආකාරයට බොහෝ සමාන හෙයින් මෙහිදී සැබෑවටම file එකක් ලෙස භාවිතා කරන ආකාරයක් user ට හැඟියනවා.

අපි බලමු OOP වල ඇති වාසි.

1. අතිරික්ත වශයෙන් ශේත ලිවීම වලකාලීමට හැකිය.(eliminate redundant code)
2. කාලය ඉතිරිකර ගත හැකිය.
3. නිරෝධනයකින්(interference) තොරව වස්තූන් ගණනාවක් පහසුවෙන් එකවර පැවතිය හැකිය.
4. ලියල ලද වැඩසටහන හෝ මොඩියුලය යාවත්කාලීන කිරීම පහසුය(easy upgrading)
5. ව්‍යාපෘති පාදක වැඩසටහන් වලදී මෘදුකාංගය කොටස් වලට බෙදීමට පහසුය

OOP වල භාවිත යෙදුම් (applications)

- Real-time systems
- Object oriented databases
- Artificial intelligences
- Expert systems
- Neural networks & Parallel programming
- Decision support and office automation systems
- CAD/CAM Systems

OOP හි මූලික සංකල්ප

- Classes and Objects
- Data encapsulation
- Data abstraction
- Interfaces
- Packages
- Inheritance
- Polymorphism
- Message communication
- Dynamic binding

අපි මේ එක එකක් ගැන ඉදිරි පාඩම වලදී පිළිවෙලින් සාකච්ඡා කරමු.

Classes සහ Objects

- **Classes**

සරලවම කිව්වොත් class එකක් යනු object එකක සැලැස්ම වගේ දෙයක්(blue print/prototype). object එකක ලක්ෂණ, ක්‍රියාකාරීත්වය යනාදිය සියල්ල තීරණය කරණු ලබන්නේ class එක විසින්. objects නිමවීමට classes යොදාගන්නවා යන්න දැන් ඔබට පැහැදිලි ඇති. එකම class එකකින් objects අවශ්‍ය තරමක් සැකසිය හැකියි. class එක සැලසුම් කිරීමේදී එහි object එකට අදාල පහත අංග ඇතුල් කෙරෙනවා.

1. **attributes/properties/state** (වස්තුවට අදාල වන ලක්ෂණ) උදා : වර්ණය, දිග, පරිමාව...
2. **constructors** (class එක මගින් object එක සෑදීමේදී object එකේ attributes(ලක්ෂණ) මොනවාදැයි ලබාදීමට භාවිතකරේ.)
3. **methods** වස්තුවේ ක්‍රියාකාරීත්වය කුමන ආකාරයෙන් වියයුතු දැයි විස්තර කෙරෙන කොටස. ඇතැම් පරිගණක භාෂාවල functions, procedures ලෙසද මෙම කොටස හැඳින්වේ.

අපි class එකකට උදාහරණයක් ලෙස Radio එකක් ගනිමු.

දැන් බලමු radio එකේ attributes සහ methods මොනවාද කියලා.

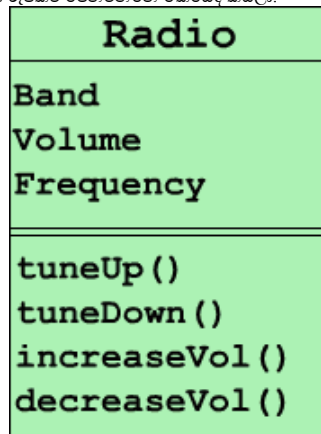
ඕනෑම radio එකකට තියෙනවා band එකක්(තරංග කලාපයක්), ඒවගේම volume(බඩද ප්‍රමාණයක්), සුසර කරන ලද සංඛ්‍යාතයක් (frequency) මේවා සියල්ල අපට radio එකේ attributes/properties ලෙස සලකන්න පුළුවන්.

ඔබ දන්නවා radio එකක් factory එකෙන් පිටවෙනකොට එයට කිසියම් පෙර සකසන ලද වින්‍යාසයක් (configuration) තිබෙනවා ඒකිව්වේ ඇතැම් radio එකක frequency එක 88.0MHz ලෙසද තවත් එකක 108.0MHz ලෙසද සැකසී එනවා. volume එකත්, band එකටත් මේ කතාව පොදුයි. මෙන්න මේක OOP වලදී ඉටුකරගැනීමට තමයි constructor එකක් භාවිතා කරන්නේ.

මිලහට radio එකක methods හඳුනා ගනිමු. ඒ කිව්වේ radio එකෙ ක්‍රියාකාරීත්වයට බලපාන දේවල්. උදාහරණයක් වීදියට radio එක සුසර කිරීම(tune) සලකමු. මෙහිදී ඉදිරියට සුසර කිරීම සහ පිටුපසට සුසර කිරීම යනාදී වශයෙන් methods දෙකක් අපට හඳුනා ගන්න පුළුවන් tuneUp සහ tuneDown වශයෙන් අපි ඒවා නම් කරමු. volume එකටත් එලෙසටම increaseVol සහ decreaseVol ලෙස methods 2ක් ගනිමු.

මේ ආකාරයට ඕනෑම object එකක methods සහ attributes හඳුනා ගත හැකියි.

දැන් අපි බලමු මෙම class එකට අයිති attributes සහ methods රූපිකව පෙන්වන්නේ කෙසේද කියලා.



දැන් අපි බලමු ඉහත class එක ජාවා තුලදී නිර්මාණය කරගන්නේ කෙසේද කියලා.

```
class ClassName {
    [Attribute Declarations]*
    [Constructors]*
    [Method Declarations]*
}
```

* යන්නෙන් අර්ථවත් කරන්නේ එකක් හෝ ඊට වැඩි ගණනක් නිබිය හැකිය යන්නයි.

අපි මේ සඳහා ජාවා කේතයක් ලියමු.

```
1. class Radio {
2.     //Attributes Declarations
3.     String Band;
4.     int Volume;
5.     float Frequency;
6.
7.     //Constructors
8.     public Radio(){
9.         Band="FM";
10.        Volume=13;
11.        Frequency=90.5f;
12.    }
13.
14.    //Method Declarations
15.    public void tuneUp(){
16.        if(Frequency < 108.0 )
17.            Frequency=Frequency+1;
18.    }
19.
20.    public void tuneDown(){
21.        if(Frequency > 88.0 )
22.            Frequency=Frequency-1;
23.    }
24.
25.    public void increaseVol(){
26.        if(Volume < 48 )
27.            Volume=Volume+2;
28.    }
29.
30.    public void decreaseVol(){
31.        if(Volume > 2 )
32.            Volume=Volume-2;
33.    }
34. }
```

දැන් ජෙලියෙන් ජෙලිය ගෙන මෙහි ක්‍රියාකාරීත්වය බලමු.
3,4,5 ජෙලි වලින් සිදුකරන්නේ class එකට අයත් attributes declare කරගැනීමයි.

8,9,10,11,12 ජෙලි අයත්වන්නේ constructor එකටයි. මෙමගින් මෙම class එක භාවිතා කර සාදාගන්නා object එකෙහි attributes/properties සඳහා අගයන්(values) initialize කිරීම සිදුකරනවා. අපි ඉදිරියේදී constructors පිළිබඳ තවදුරටත් සාකච්ඡා කිරීමට බලාපොරොත්තු වෙනවා.

15 වන ජෙලියේ සිට පහලට class එකට අයත් methods පිළිබඳ විස්තර කිරීම සිදුකෙරෙනවා. මෙහිදී භාවිතාවී ඇති සෑම method නමකම මුලට public void ලෙසින් යෙදීමට හේතුව අපි ඉදිරියේදී සාකච්ඡා කරන method සම්බන්ධ පාඩමේදී බලමු.

මෙම පාඩමේදී class එකක් තුල attributes,constructors හා methods යෙදෙන ආකාරය දැනගැනීම ප්‍රමාණවත්.

අපි ඊලඟ පාඩමෙන් object, constructors හා methods පිළිබඳව බලමු.

Java වස්තු පාදක ක්‍රමලේඛනය II (objects)

Sunday, May 16, 2010 by Kaniskha Dilshan

- **Objects**

සැසු: මෙම පාඩම [කලින් පාඩම\(Java classes\)](#) සමග සම්බන්ධ බව සලකන්න.

Object එකක් යනු data සහ method වල එකතුවක් ලෙස හඳුන්වන්න පුලුවන්. සැබෑ ලෝකයේ වස්තූන් Java තුල නිරූපණය කරන්නේ objects භාවිතයෙන්. classes තමයි object එකේ design එක පිළිබඳ තීරණය කරන්නේ.

class එක භාවිත කර objects සෑදූ පසු ඒවා එක එකක් වෙන වෙනම පරිගණකයේ ප්‍රධාන මතකයේ(RAM) තැන්පත් වීම සිදුවනවා. ඉන්පසු අදාළ කාර්යය ඉටුකර ගැනීම සඳහා අපට ඒවා යොදාගන්න පුලුවන්.

අපි බලමු පසුගිය පාඩමේදී අපි යොදාගත් Radio class එක භාවිතයෙන් objects සකසා ගන්නා ආකාරය.
ඔබගේ පහසුව සඳහා මම Radio class එකේ කේතය නැවත මෙම පාඩමේ ඉදිරිපත් කරනවා.

Radio class

```
1. class Radio {
2.     //Attributes Declarations
3.     String Band;
4.     int Volume;
5.     float Frequency;
6.
7.     //Constructors
8.     public Radio(){
9.         Band="FM";
10.        Volume=13;
11.        Frequency=90.5f;
12.    }
13.
14.    //Methods
15.    public void tuneUp(){
16.        if(Frequency < 108.0 )
17.            Frequency=Frequency+1;
18.    }
19.
20.    public void tuneDown(){
21.        if(Frequency > 88.0 )
22.            Frequency=Frequency-1;
23.    }
24.
25.    public void increaseVol(){
26.        if(Volume < 48 )
27.            Volume=Volume+2;
28.    }
29.
30.    public void decreaseVol(){
31.        if(Volume > 2 )
32.            Volume=Volume-2;
33.    }
34. }
```

අපි දැන් බලමු class එකකින් objects සාදාගැනීමට අදාළ Java syntax මොනවාද කියලා.උදාහරණ සඳහා ඉහත Radio class එකම යොදා ගනිමු.

class එකකින් object එකක් සාදා ගැනීම සඳහා **new** Keyword එක භාවිතා කෙරෙනවා.

Syntax 01:

```
1.    ClassType ObjectName;
```

2. `ObjectName=new ClassType(); //default constructor`
3. `//or`
4. `ClassType ObjectName;`
5. `ObjectName=new ClassType(parameter-list);`

Example :

1. `Radio rad1;`
2. `rad1=new Radio();`

Syntax 02:

1. `ClassType ObjectName=new ClassType();`
2. `//or`
3. `ClassType ObjectName=new ClassType(parameter-list);`

Example :

1. `Radio rad1=new Radio();`

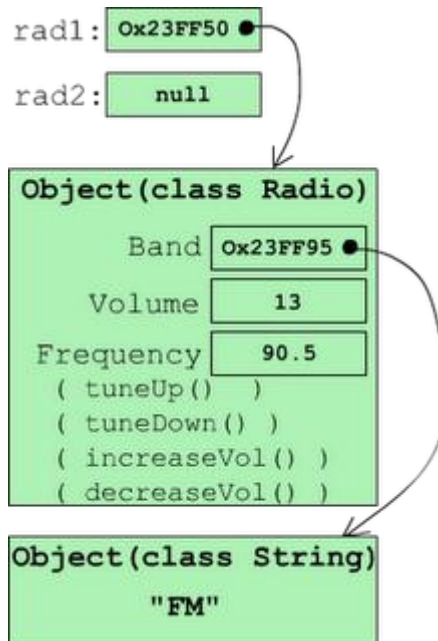
දැන් අපි බලමු අපි සකසාගත් object එක පරිගණකයේ ප්‍රධාන මතකයේ(RAM) තැන්පත් වීම සිදුවන්නේ කොහොමද කියලා.

මුනිත්ම **new** keyword එක ගැන බලමු.

new keyword එක මගින් සිදුවන්නේ දී ඇති class type එකෙන් object එකක් RAM එක තුළ සාදා එයට reference එකක් (යොමුවක්) අදාළ variable එක වෙත යැවීමයි. ඉහත උදාහරණය අනුව ගතහොත්, unique(අනන්‍ය) Radio object එකක් RAM එක තුළ සාදා එයට reference එකක් rad1 නමැති variable එකට assign කිරීම සිදුකරයි. මෙහිදී ඔබ තේරුම් ගත යුතු කරුණ වන්නේ ඇති Java හි භාවිතා වන variable දරාගෙන(hold) සිටින්නේ object එකක් නොව object එක සඳහා reference එකකි. තව දුරටත් පැහැදිලි කරන්නේ නම් Java variable එකක් තුළ ඇත්තේ object එක RAM එක තුළ පවතින ස්ථානයට අදාළ memory address එකකි. memory address එකක් සඳහා උදාහරණ : 0x23FF50 එනම් object එක හඳුනා ගැනෙනුයේ අදාළ memory address එකෙනි.

දැන් අපි බලමු object එක කොහොමද RAM එක තුළ තිබෙන්නේ කියලා.

object එකක් RAM එකේ heap නමැති විශේෂ කලාපය තුළ වසිනවා. ඉහත rad1 නමැති object එක සෑදීමෙන් පසුව rad1 object එක RAM එකේ පවතින ආකාරය පහත දැක්වෙන සටහන අනුව බලමු.



දැන් object එකක් RAM එකේ තිබෙන ආකාරය ගැන පැහැදිලි ඇතියි සිතනවා.

ඔබ දකින්න ඇති rad2 කියලා තිබුන Radio object එකේ තිබුන **null** කියලා value එකක්. **null** කියන්නේ Java තුළ භාවිතාවන literal value එකක් ඇතැම් තැන් වල manifest constant එකක් කියලත් හඳුන්වනවා. නමුත් මෙය ජාවා keyword එකක් නොවෙයි.

හොඳයි මොකක්ද මේ **null** value එකෙන් වෙන්නේ?

මෙමගින් object එකක් memory location එකක් refer නොකරන බව එනම් එය අභිභූතය බව දක්වනවා. rad2 යනු **null** object එකක් නිසා එය කිසිදු memory location එකක් සමග බැඳී නැහැ කිව්වොත් නිවැරදියි.

ජාවා පරිගණක භාෂාවේ ඇති සුවිශේෂ වාසියක් වන්නේ සාදන ලද objects සඳහා වෙන් කරන ලද memory ප්‍රමාණයන් නැවත නිදහස් කිරීම සඳහා වෙහෙස වීමට අවශ්‍ය නොවීමයි. එය සිදුකිරීම සඳහා JVM එක මගින් ක්‍රියාත්මක වන සුවිශේෂ thread එකක් වන "Garbage Collector" බැඳී සිටිනවා. අපගේ වැඩසටහන මගින් සකසන ලද කිසියම් object එකක් තවදුරටත් භාවිතා නොවන තත්වයට පත්වූ විට "Garbage Collector" විසින් එම object එක සඳහා වෙන් කරන ලද memory ප්‍රමාණය නිදහස් කිරීම සිදුකරනවා. මෙහි ඇති වාසිය සරළ ජාවා වැඩසටහන් වලදී එතරම් නොපෙනෙනද සංකීර්ණ data structures භාවිතාවන වැඩසටහන් වලදී හොඳින් කැපී පෙනෙනවා. C/C++ යොදාගෙන එවැනි වැඩසටහන් ලිවීමේදී අතිවෘද්ධීන් වෙන්කරන ලද (allocate) memory ප්‍රමාණය තව දුරටත් අවශ්‍ය නොවන තත්වයට පත්වූ විට release කළ යුතුයි. එසේ release නොකිරීම නිසා ඇතිවන තත්වය "memory leak" එකක් ලෙස හැඳින්වේ. එය පරිගණක වැඩසටහනක ඇති දෝෂ තත්වයකි. ක්‍රියාකාරී ජාවා වැඩසටහනක process එකට අදාළ heap කොටස ගැන පහත රූප සටහන ආධාරයෙන් නිරීක්ෂණය කරන්න.

0x000b5068	0x000b5060	536	0x000b0000	0
0x000b5288	0x000b5280	96	0x000b0000	0
0x000b5340	0x000b5338	136	0x000b0000	0
0x000b53e8	0x000b53e0	104	0x000b0000	0
0x001b0688	0x001b0680	6144	0x001b0000	0

000B5288	00 00 00 00 61 76 61 5F 6A 61 76 61 5F 73 65 63	...
000B5298	75 72 69 74 79 5F 41 63 63 65 73 73 43 6F 6E 74	uri
000B52A8	72 6F 6C 6C 65 72 5F 64 6F 50 72 69 76 69 6C 65	rol
000B52B8	67 65 64 5F 5F 4C 6A 61 76 61 5F 73 65 63 75 72	ged
000B52C8	69 74 79 5F 50 72 69 76 69 6C 65 67 65 64 41 63	ity
000B52D8	74 69 6F 6E 5F 32 40 31 32 00 00 00 5C 00 4D 00	tio

මෙම වැඩසටහන download කරගැනීමට අවශ්‍ය නම් මෙම link එක යොදාගන්න
අපි ඊලඟ පාඩමෙන් constructors සහ methods ගැන අධ්‍යයනය කරමු.

ඡායා වස්තු පාදක ක්‍රමලේඛනය III (Constructors)

Wednesday, May 19, 2010 by Kaniskha Dilshan

- Constructors



constructor මගින් සිදුකරනුයේ එය අයත් class එකට අදාලව සාදන object එක initialize කිරීමයි. දැනටමත් ඔබ දන්නවා class එකකින් කොහොමද object එකක් සාදාගන්නේ කියලා. (පසුගිය පාඩම). මේ සඳහා **new** keyword එක භාවිතාවන බවත් ඔබ දන්නවා ඇති.

අපි බලමු constructor එකක ලක්ෂණ මොනවාද කියලා.

- ▶ constructor එකක නම එය අයත් class එකේම නමම වනවා. උදා: Radio class එකේ constructor එකේ නමත් Radio ම වනවා.
- ▶ arguments එකක් හෝ කිහිපයක් තිබිය හැකියි. උදා: Radio (String band, float frequency)
- ▶ return values ගැන කිසිදු සඳහනක් නොමැත. (methods ගැන සාකච්ඡා කිරීමේදී return values ගැන බලමු)
- ▶ constructor එකක් හෝ කීපයක් පැවතිය හැකිය.

constructors එකක කාඨි භාරය කුමක්ද?

- local variable වල ආරම්භක අගය සැකසීම.(initialize)
- කීප ආකාරයකට object එක සෑදීමට යොදාගත හැකිවීම

Syntax:

1. public ConstructorName(argument-list){
2. //constructor body
3. }

උදාහරණයක් ලෙසට අපි Circle එකක් ගනිමු.

Circle.java

```
1. class Circle {
2.     int x,y,radius;
3.     //Constructor 1
4.     public Circle(int x,int y,int r){
5.         this.x = x;
6.         this.y = y;
7.         radius=r;
8.     }
9.     //Constructor 2
10.    public Circle(int x,int y){
11.        this.x = x;
12.        this.y = y;
13.        radius=1;
14.    }
15. }
```

දැන් අපි බලමු Circle class එකෙන් Circle object එකක් සෑදීමේදී constructor එක වැදගත් වන ආකාරය. අපි ඉහත class එකේ අන්තර්ගත කරන ලද constructor භාවිතාකර object 2ක් සාදමු.

```
1. //using constructor 1
2. Circle c1=new Circle(12,34,7);
3. //using constructor 2
4. Circle c2=new Circle(14,47);
```

c1 object එක සෑදීමේදී යෙදෙන්නේ constructor 1 එකයි. එම constructor එකෙහිදී සිදුවන්නේ x, y හා r ලෙසින් arguments 3ක් ලබාගෙන c1 object එකෙහි class variables ලෙස පවතින x, y, හා radius variables initialize කිරීමයි.

c2 object එක සෑදීමේදී යෙදෙන්නේ constructor 2 එකයි. එහිදී arguments ලෙස ලබා ගනුයේ x හා y පමණි. radius එක 1 ලෙසට initialize කිරීම මෙම constructor එකෙන් සිදුකරයි. එනම් මෙම constructor එක(constructor 2) භාවිතයෙන් සාදන සෑම object එකකම radius එකෙහි initial value එක 1 වේ.

දැන් අපි බලමු **this keyword** එකෙහි භාවිතය පිළිබඳව

this keyword එක භාවිතා වන්නේ කිසියම් object instance එකකට අයත් දෑ එම object එක තුලදී refer කිරීමටයි.

මෙහිදී this keyword එක භාවිතා වන්නේ class variables හා arguments refer කිරීමේදී ඇතිවන ගැටලුවෙන් මිදීමටයි. class variables යනු සමස්ත class එකටම බලපවත්වන ලෙස පවතින variables ය. ඉහත **Circle** class එකෙහි x, y හා radius යනු Circle class එකෙහි class variables වේ. constructor 1 හි තවත් x හා y ලෙසින් arguments 2ක් ගෙන ඇති බව ඔබට පෙනෙනවා ඇත. එම නමුත් එක x හා y විවලායන් බලපවත්වනුයේ/වලංගු වනුයේ constructor body එක තුලදී පමණි.

this.x ලෙසින් අප refer කරනුයේ class object එකට අයත් x අගයයි. අනෙක් x එක argument එකයි. එනම් **this.x = x** යන්නෙහි අර්ථය වන්නේ argument එකක් ලෙස සපයන x හි අගය object එකට අයත් x වෙත ලබාදීමයි.

ජාවා වස්තු පාදක ක්‍රමලේඛනය III (Methods)

Thursday, May 20, 2010 by Kaniskha Dilshan

Methods



Method එකක් යනු function/procedure සඳහා ලබාදී ඇති OOP term එකයි. methods වල වැදගත්කම කිහිපයක් බලමු.

- ▶ වැඩසටහනක් modules වලට කැඩීමේදී උපයෝගී කරගනී.
- ▶ Code reusability සංකල්පය එනම් කේත නැවත නැවත, අදාළ ස්ථානයන්හිදී භාවිතා කිරීමට යොදාගත හැක.
- ▶ Method එකක් ක්‍රියාත්මක(execute) වනුයේ එය call කල විට පමණි (explicitly invoked).

Method Declaration

Method එකක් සෑම විටම පැවතිය යුත්තේ class එකක් තුලයි. method එකක් declare කිරීමෙන් පසු අපට අවශ්‍ය විටක එය invoke කල හැකිය. එතෙක් එය තුල ඇති උපදෙස් ක්‍රියාත්මක නොවේ. පහත syntax එකට අනුව method එකක් declare කිරීම සිදුකෙරේ.

Syntax:

```
modifiers returnType methodName (parameter-list) {
    /*
```

```

    * Method body (block)
    */
    return expression;
}

```

දැන් අපි method header එක පිළිබඳව සලකමු.

■ modifiers

මෙයට method scope යයිද කියනවා. මෙමගින් method එක access කිරීම පිළිබඳ තීරණ ගනු ලබනවා.

මෙම access modifiers වනුයේ public, private, protected, default/friendly යන ඒවායි. මේවා ගැන සවිස්තර වශයෙන් අපි ඉදිරියේදී සාකච්ඡා කරමු.

■ returnType

method එකෙන් එය call කල ස්ථානයට යවන(return) පණිවුඩය කුමන දත්ත ආකාරයක් ඇති එකක්ද යන්න මෙමගින් සඳහන් කෙරේ.

■ method එක invoke කරනවිට method එකට ලබාදෙන දත්ත parameter-list එක තුල අන්තර්ගත වේ.

■ method body

method එකෙන් සිදුකරන කාර්යයට අදාල කේත මේතුල අඩංගුවේ.

■ return expression

method එකෙන් එය call කල ස්ථානයට පණිවුඩය යැවීම මෙමගින් සිදුකෙරේ. මෙම statement එක execute වී අවසන් වූ වහාම method එකෙහි execute වීම අවසන්වේ.

Note : සෑම method එකක්ම values return කිරීම අනිවාර්ය නොවේ. මෙසේ values return නොකරන method වලදී modifier එක ලෙස **void** යන්න භාවිතාවේ.

Method invoke/call කිරීම

මෙය method එක සඳහා values pass කිරීම ලෙසද හඳුනවනු ලැබේ.

syntax:

methodName (argument-list)

argument-list එකෙහි ඇති arguments හරියටම method එකේ parameter-list එකෙහි ඇති parameters වලට අනුරූප විය යුතුය.

variable pass කරන්නේ නම් මෙම arguments අගයන් සහිත තාත්ත්වික විචල්‍යයන්(real variables) විය යුතුය. මේ නිසා මෙම arguments pass කිරීමට පෙර එම variables initialize කර තිබිය යුතුය.

► class එක තුලදී method call කිරීම

මෙහිදී කෙලින්ම method එකේ නම භාවිතාකරමින් invoke කල හැක. අවශ්‍ය නම් this keyword එකද භාවිතාකල හැකිය.

methodName ()

//or

this.methodName ()

► class එකෙන් පිටතදී method call කිරීම

මෙහිදී method එක අයත් object එකේ නමද යොදාගැනේ

objectName .methodName ()

► static method එකක් නම්

static keyword එක ගැන අපි ඉදිරි පාඩමකදී විස්තරාත්මකව බලමු. දැනට static method එකක් call කරන ආකාරය මතක තබා ගන්න.

className .methodName ()

ex:

```

1.    Math.random();

```

දැන් අපි බලමු methods පවතින සරල ආකාර මොනවාද කියලා

► return type එකක් පවතින සහ parameters නැති

උදා:

```

1.    int getName(){
2.        return name;
3.    }

```

► return type එකක් පවතින සහ parameters ඇති

```

1.    int sum(int i,int j){
2.        return(i+j);
3.    }

```

► return type එකක් නොපවතින(void) සහ parameters නැති

```

1.    void showDetails(){
2.        System.out.println("Name: " + this.name);
3.    }

```

► return type එකක් නොපවතින(void) සහ parameters ඇති

```

1.    void showMax(int a, int b){
2.        if(a>b)
3.            System.out.println(a);
4.        else
5.            System.out.println(b);
6.    }

```

මෙම ආකාරවලට අමතරව methods පැවතිය හැකි තවත් ආකාර කිහිපයක් පවතිනවා. අපි ඒවා වෙනමම පාඩමකදී සාකච්ඡා කරමු.

දැන් අපි method හි ක්‍රියාකාරීත්වය තවත් හොඳින් අවබෝධ කරගැනීම සඳහා ජාවා වැඩසටහනක් ලියමු.
මේ සඳහා [කුඩින් පාඩමේදී](#) ගත් උදාහරණයම methods යොදා දීර්ඝ කරමු.

Circle.java

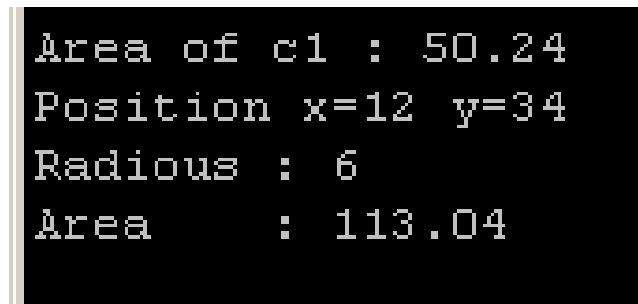
```
1.  class Circle {
2.      int x,y,radius;
3.      //Constructor 1
4.      public Circle(int x,int y,int r){
5.          this.x = x;
6.          this.y = y;
7.          radius= r;
8.      }
9.      //Constructor 2
10.     public Circle(int x,int y){
11.         this.x = x;
12.         this.y = y;
13.         radius=1;
14.     }
15.
16.     //Method 1
17.     public float getArea(){
18.         return (3.14f*radius*radius);
19.     }
20.     //method 2
21.     public void setRadius(int r){
22.         radius=r;
23.     }
24.     //method 3
25.     public void showDetails(){
26.         System.out.println("Position x=" +x+" y="+y );
27.         System.out.println("Radius : " + radius );
28.         System.out.println("Area   : " + getArea() );
29.     }
30. }
```

දැන් බලමු එම circle class එකෙන් සාදන object වලට අයත් methods භාවිතා කරන ආකාරය.
මේ සඳහා ලියන ලද සරල java වැඩසටහනක් පහත දැක්වේ.

CircleDemo.java

```
1.  public class CircleDemo {
2.      public static void main(String srgs[]){
3.          /*create a circle object
4.          *x=12 , y=34 , radius=4
5.          */
6.          Circle c1=new Circle(12,34,4);
7.          float c1_area=c1.getArea();
8.          System.out.println("Area of c1 : " + c1_area);
9.
10.         //change the radius of c1
11.         c1.setRadius(6);
12.         //invoke the showDetails method of c1
13.         c1.showDetails();
14.
15.     }
16. }
```

Output :



```
Area of c1 : 50.24
Position x=12 y=34
Radius : 6
Area : 113.04
```

මෙම පාඩමෙන් ලබාගත් දැනුම උපයෝගී කරගනිමින් ඉහත වැඩසටහන ක්‍රියාත්මක වන ආකාරය දැන් ඔබට තේරුම් ගත හැකිය. හැටලු වෙනතත් comment එකක් ලෙස හෝ ඉදිරිපත් කරන්න.

අපි මීලඟ පාඩමෙන් access modifiers ගැන සාකච්ඡා කරමු.

ජාවා වස්තු පාදක ක්‍රමලේඛනය IV (Access Modifiers)

by Kaniskha Dilshan

මීට පෙර පාඩමෙන් අපි සාකච්ඡා කළේ [methods](#) ගැනයි. අපි එහිදී access modifiers සඳහා සුලු හැඳින්වීමක් දුන්නා. අපි මෙම පාඩමෙන් modifiers ගැන විස්තර සහිතව අධ්‍යයනය කරමු. Inheritance , Encapsulation වැනි OOP Concept ක්‍රියාවට නැංවීම (implement) සඳහා access modifiers ඉතා වැදගත් මෙහෙවරක් ඉටුකරනවා. අපි පහත වගුව තුළින් access modifiers ගැන අධ්‍යයනයක යෙදෙමු.

Access modifier	this(class)	subclass	package	general
public	•	•	•	•
protected	•	X	X	X
default	•	X	•	X
private	•	X	X	X

දැන් අපි මේ පිළිබඳව විස්තරාත්මකව බලමු.

■ public

මෙලෙස declare කරන ලද members සඳහා ජාවා වැඩසටහන තුළ එය අයත් object එක refer කරන ඕනෑම තැනකදී ලඟ විය හැක.

■ protected

මේ අයුරින් declare කරන ලද methods හා variables භාවිතා කළ හැක්කේ එම class එකෙහි sub class / child class වලට පමණි. මෙය Inheritance පාඩමේදී වඩාත් පැහැදිලි වනු ඇත.

■ default

යම් member කෙනෙක් සඳහා කිසිදු modifier එකක් යොදා නොමැති විට එයට යෙදෙනුයේ default modifier එකයි. default members භාවිතා කළ හැක්කේ එම class එක තුළදී සහ එම class එක අයත් package එක තුළදී පමණි.

■ private

සියලු private members access කළ හැක්කේ එය අයත් class එක තුළදීම පමණි.

සැසු: සාමාන්‍යයෙන් **Object Oriented** ක්‍රමලේඛනයක් ලිවීමේදී අනුගමනය කරන දෙයක් නම්යි **data** නොහොත් **instance variables**, **private** ලෙස **declare** කිරීම සහ **methods**, **public** ලෙස **declare** කිරීම. **public data** භාවිතා කිරීම හොඳ වස්තු පාදක ක්‍රමලේඛනයක ලක්ෂණයක් නොවේ!.

දැන් අපි **private** සහ **public** modifiers ගැන අධ්‍යයනය කිරීම සඳහා සරල ජාවා වැඩසටහනක් ලියමු.

මේ සඳහා අපි **Employee** කෙනෙකු උදාහරණයට ගනිමු.

Employee.java

```
1. class Employee {
2.     //instance variables
3.     public String Name; // Name is a public member
4.     private float HourlyRate; //private member
5.     private float HoursWorked; //private member
6.
7.     //constructor
8.     public Employee(String name,float hr,float hw){
9.         Name=name;
10.        HourlyRate=hr;
11.        HoursWorked=hw;
12.    }
13.
14.    //public method
15.    public void setHourlyRate(float hr){
16.        HourlyRate=hr;
17.    }
18.    //private method
19.    private float getTotalSal(){
20.        return (HourlyRate*HoursWorked);
21.    }
22.    //public method
23.    public void showDetails(){
24.        System.out.println("Employee Name : " + Name);
25.        System.out.println("Total Salary : " + getTotalSal() );
26.        System.out.println("-----");
27.    }
28. } //end of the Employee class
```

■දැන් අපි Employee class එක යොදාගෙන පහත කේතය execute කර ලැබෙන ප්‍රතිඵලය කුමක්දැයි බලමු.

```
1. Employee emp1=new Employee("Prasanna",2300.75f,40.0f);
2. emp1.showDetails();
3. emp1.Name="Nimal"; //access the public variable of emp1
4. emp1.showDetails();
```

1 සහ 2 ඡේද

මෙහිදී **Employee** object එකක් සාදා මුලින්ම එහි public member කෙනෙකු වන **showDetails()** method එක invoke කරයි.

3 ඡේදය

මෙහිදී ඉහත සාදාගත් **emp1** object එකට අයත් public member වන Name විචල්‍යය access කර එහි name එක Nimal ලෙසට වෙනස් කරනු ලැබේ.

4 ඡේදය

මෙහිදී නැවතත් **emp1** object එකෙහි **showDetails()** method එක invoke කරයි. දැන් console එකෙහි දිස්විය යුත්තේ Name එක Nimal ලෙසටය.

Output:

```
Employee Name : Prasanna
Total Salary  : 92030.0
-----
Employee Name : Nimal
Total Salary  : 92030.0
-----
```

■ මීලඟට private modifier එක අධ්‍යයනය කිරීමට පහත කේතය execute කරමු.

1. Employee emp1=new Employee("Prasanna",2300.75f,40.0f);
2. emp1.HourlyRate=1500.00f;

මෙහිදී **emp1** ගේ **private** modifier එකක් වන HourlyRate විචල්‍යය අප access කිරීමට යයි. එම නිසා මෙහිදී compile කිරීමේදී පහත දෝශ පණිවුඩය පෙන්වයි.

```
PMod.java:42: HourlyRate has private access in Employee
    emp1.HourlyRate=1500.00f;
        ^
1 error
```

අපට නිවැරදිව HourlyRate එක වෙනස් කිරීමට කලයුතු වන්නේ අප Employee class එකෙහි අන්තර්ගත කොට ඇති **setHourlyRate()** නම් public method එක භාවිතා කිරීමයි. මේ සඳහා කේතය පහත පරිදි වේ.

1. Employee emp1=new Employee("Prasanna",2300.75f,40.0f);
2. emp1.showDetails(); // before changing the HourlyRate
3. emp1.setHourlyRate(1500.00f);
4. emp1.showDetails(); // after changing the HourlyRate

Output:

```
Employee Name : Prasanna
Total Salary  : 92030.0
-----
Employee Name : Prasanna
Total Salary  : 60000.0
-----
```

මේ ආකාරයෙන්ම Employee class එකට අයත් **getTotalSal()** නම් private method එක class එක තුලදීම භාවිතා කර තිබෙන ආකාරය line 25 දෙස බලන විට ඔබට පෙනී යනු ඇත.

අනෙක් access modifiers පිළිබඳව අපි ඉදිරියේදී සාකච්ඡා කරන OO Concept වලදී පුලුල්ව අධ්‍යයනය කිරීමට පුලුවන්.

මම මේ සඳහා යොදාගත් උදාහරණ භාගත කර ගැනීමට පහත සබැඳිය භාවිතා කරන්න.

[OOP සඳහා අභ්‍යාස I](#)

Saturday, May 22, 2010 by Kaniskha Dilshan

අපි මීට පෙර පාඩම් වලදී ඉගෙනගත් Java OOP සංකල්ප තවත් හොඳින් අවබෝධ කරගැනීම සඳහා අපි අභ්‍යාස කිහිපයක් සාකච්ඡා කරමු. මෙහිදී මම හැකිතාක් ප්‍රායෝගික උදාහරණ ඉදිරිපත් කිරීමට බලාපොරොත්තු වනවා. අභ්‍යාසයේ යෙදෙන තරමට අපිට ජාවා වල සංකල්ප අවබෝධ කරගැනීමට පහසුයි.

අපි මුලින්ම සරල උදාහරණයකින් පටන්ගනිමු. මෙහිදී ඉදිරිපත් කර ඇති කේතයන් ඔබ ඉගෙනගත් OOP concept සමග ගලපමින් අවබෝධ කරගැනීමට උත්සාහ කරන්න.

Exercise1

```

1.  /**
2.   *Author : Kanishka Dilshan
3.   *Blog  : http://javaclass.blogspot.com
4.   *Purpose: Demonstrate some applications of OOP
5.   */
6.
7.   class Car {
8.       private String model,color;
9.       private int enginePower,speed;
10.
11.      public Car(String model,String color,int enginePower){
12.          this.model=model;
13.          this.color=color;
14.          this.enginePower=enginePower;
15.          speed=0;
16.      }
17.
18.      public void accelerate(){
19.          //let's assume the maximum speed is 320
20.          if(speed < 320){
21.              speed+=10; //increment speed by 10
22.          }
23.      }
24.      public void applyBreaks(){
25.          if(speed > 10){
26.              speed-=10; //decrement speed by 10
27.          }
28.      }
29.      public void changeColor(String NewColor){
30.          this.color=NewColor;
31.      }
32.      public void showDetails(){
33.          System.out.println("-----");
34.          System.out.println("Car Model   : " + model);
35.          System.out.println("Engine Power : " + enginePower);
36.          System.out.println("Color       : " + color);
37.          System.out.println("Current Speed: " + speed);
38.          System.out.println("-----");
39.      }
40.      public String getModel(){
41.          return model;
42.      }
43.  }
44.
45.  public class Exercise1{
46.      public static void main(String args[]){
47.          //create a new Car object
48.          Car c1=new Car("Audi A4","Blue",2000);
49.          //accelerate the car
50.          for(int i=0;i < 120;i++){
51.              c1.accelerate();
52.          }
53.          //show details of the car
54.          c1.showDetails();
55.          //apply breaks on the car(3 times)
56.          c1.applyBreaks();
57.          c1.applyBreaks();
58.          c1.applyBreaks();
59.          //change color
60.          c1.changeColor("Black");
61.          //show details of the car again
62.          c1.showDetails();
63.
64.          //create another car object
65.          Car c2=new Car("Mazda 3 Sedan","Red",1800);
66.          //.....
67.      }
68.  }

```

Output :

```

-----
Car Model   : Audi A4
Engine Power : 2000
Color       : Blue
Current Speed: 320
-----

Car Model   : Audi A4
Engine Power : 2000
Color       : Black
Current Speed: 290
-----

```

දැන් අපි ඉහත ලියන ලද ජාවා වැඩසටහන විස්තරාත්මක වශයෙන් සාකච්ඡා කරමු.

Line 07 මගින් Car class එකේ ආරම්භය සනිටුහන් කෙරෙනවා. එහි සියලුම class variables වලට යොදා ඇත්තේ private modifier එක බව ඔබට පෙනෙනවා ඇති.(line 8,line 9). අපි access modifiers ගැන සාකච්ඡා කරද්දී ඔබට මතක ඇති මම එහිදී සඳහන් කලා instance variables,private ලෙසත් methods , public ලෙසත් යෙදීම හොඳ වස්තු පාදක ක්‍රමලේඛනයක ලක්ෂණයක් බව.

ඉන්පසුව ඇත්තේ constructor එකයි (line 11) එහිදී අපි speed එක 0 ලෙසත් අනෙකුත් class variables සඳහා constructor එකෙන් ලබාගන්නා parameters ද ආදේශ කර තිබෙනවා. මෙහිදී constructor එකේ parameter එකක් ලෙස speed එක ගෙන නැහැ. නමුත් constructor එක තුලදී speed variable එක සුදුසු value එකක් යොදා initialize කර තිබෙනවා.

ඉන්පසු ඇත්තේ methods ය. අපි methods පාඩමේදී ලබාගත් දැණුම භාවිතයෙන් ඔබට මෙය පහසුවෙන් අවබෝධ කරගත හැකිවිය යුතුය.

Car class එක ලියා අවසන්වූ පසු main method එක අයත් Exercise1 class එක ආරම්භවේ. මෙහිදී අප සකස්කරගත් Car class එක භාවිතා කර object සාදා ඒවා භාවිතා කිරීම සිදු කෙරේ. ජාවා වැඩසටහන තුල යොදා ඇති comments වලින්ද එහිදී සිදුවන්නේ කුමක්ද යන්න තේරුම් ගැනීමට හැක.

Exercise2

```
1.  /**
2.   *Class Name : Excerice2java
3.   *Author    : Kanishka Dilshan
4.   *Blog      : http://javaxclass.blogspot.com
5.   *Purpose   : Demonstrate some applications of OOP
6.   */
7.   class Line {
8.       private int x1,y1,x2,y2;
9.
10.      public Line(int x1,int y1,int x2,int y2){
11.          this.x1=x1;
12.          this.y1=y1;
13.          this.x2=x2;
14.          this.y2=y2;
15.      }
16.
17.      public double getLength(){
18.          double length=Math.sqrt(Math.pow((x1-x2),2)+Math.pow((y1-y2),2));
19.          return length;
20.      }
21.  }
22.
23.  public class Exercise2{
24.      public static void main(String args[]){
25.          Line line1=new Line(-3,6,10,8);
26.          double len=line1.getLength();
27.          System.out.println(len);
28.      }
29.  }
```

අපි ඊළඟ පාඩමෙන් මීට වඩා සංකීර්ණ උදාහරණ කිහිපයක් බලමු.

OOP සඳහා අභ්‍යාස II කොටස

Sunday, May 23, 2010 by Kanishka Dilshan

අපි කලින් පාඩමෙන් සරල OOP සඳහා උදාහරණ දෙකක් සාකච්ඡා කලා. අපි මේ පාඩමේදී බලමු මඳක් සංකීර්ණ OOP examples කිහිපයක්. අපි මෙතෙක් උගත් Object Oriented සංකල්ප වලින් පමණක් උදාහරණ සපයා ඇති ඇතිබව සලකන්න. අපි ඉදිරියේදී ජාවාහි තවත් වැදගත් Object Oriented සංකල්ප සාකච්ඡා කරමු. එහිදී ඒවාට අදාල උදාහරණ සුදුසු පරිදි ඉදිරිපත් කෙරෙනු ඇත.

අපි මෙහිදී objects අතර සම්බන්ධතාවයක් පෙන්විය හැකි උදාහරණයක් සලකා බලමු. objects ලෙස Point හා Line (ලක්ෂ්‍යයක් හා සරල රේඛාවක්) ගනිමු. මෙහිදී අපි එක එකක් class සඳහා වෙන වෙනම java file ලියා තිබෙනවා. පාඩම අවසානයේ එම මූලාශ්‍ර කේත අමුණා තිබෙනවා.

Point.java

```
1.  /**
2.   *Class Name : Point
3.   *Author    : Kanishka Dilshan
4.   *Bolg      : http://javaxclass.blogspot.com
5.   *Purpose   : Demonstrate relationships between objects
6.   */
7.   public class Point {
8.       private int x,y;
9.
10.      public Point(int x,int y){
11.          this.x=x;
12.          this.y=y;
13.      }
14.
15.      public void setX(int x){
16.          this.x=x;
```

```

17.     }
18.
19.     public void setY(int y){
20.         this.y=y;
21.     }
22.
23.     public int getX(){
24.         return x;
25.     }
26.
27.     public int getY(){
28.         return y;
29.     }
30.
31.     public double distance(){
32.         //calculate and return distance from the origin
33.         double dis=Math.sqrt(Math.sqrt(Math.pow(x,2)+Math.pow(y,2)));
34.         return dis;
35.     }
36.
37.     }

```

දැන් ඔබට Point class එකේ ඡායා කේතය කියවා අවබෝධ කරගැනීමට හැකිවිය යුතුය. ගැටලුවක් ඇත්නම් [පෙර OOP පාඩම්](#) නැවත කියවා බලන්න. **Math.pow()** හා **Math.sqrt()** යනු ඡායාගි inbuilt methods ය. **pow()** method එක මගින් යම් සංඛ්‍යාවක බලයක්ද **sqrt()** method එක මගින් යම් සංඛ්‍යාවක වර්ගමූලයද ලබාදේ. දැන් අපි අපි ඉහත ලියන ලද Point class එක සම්බන්ධ කරගෙන Line (සරල රේඛාවක්) සඳහා class එකක් ලියමු. source code එක කියවීමේදී statements වලට අදාලව යොදා ඇති comments ද කියවන්න. එමගින්ද අදාල කොටස ගැන හොඳ අවබෝධයක් ලබාගත හැකිවේවි.

Line.java

```

1.     /**
2.      *Class Name : Line
3.      *Author    : Kanishka Dilshan
4.      *Bolg     : http://javaxclass.blogspot.com
5.      *Purpose   : Demonstrate relationships between objects
6.      */
7.     public class Line {
8.         private Point p1,p2;
9.         /**
10.          *p1 and p2 are Point objects
11.          *we have already created the Point class
12.          */
13.         public Line(Point p1,Point p2){
14.             this.p1=p1;
15.             this.p2=p2;
16.         }
17.
18.         public double getLength(){
19.             int x1=p1.getX();
20.             int x2=p2.getX();
21.             int y1=p1.getY();
22.             int y2=p2.getY();
23.             double len=Math.sqrt(Math.pow((x1-x2),2)+Math.pow((y1-y2),2));
24.             return len;
25.         }
26.
27.         public void showDetails(){
28.             System.out.println("x1 : " + p1.getX() + " y1 : " + p1.getY());
29.             System.out.println("x2 : " + p2.getX() + " y2 : " + p2.getY());
30.             System.out.println("-----");
31.         }
32.
33.     }

```

ඔබ දකින්නට ඇති Line class එකේ 8 වන පේලියේදී අපි p1 හා p2 ලෙසින් **Point** object 2ක් සාදාගෙන තිබෙනවා. එම **Point** type එක වක්‍රයේ අපි ඉහතින් සකස්කර ගත් **Point** class එකයි.

අපි Line class එකේ constructor එකේදී (13පේලිය) **Point** parameters 2ක් ගෙන තිබෙනවා.අපි දැන් ඉහත class 2ම යෙදෙන පරිදි ඡායා වැඩසටහනක් ලියමු.

```

1.     /**
2.      *Class Name : Line
3.      *Author    : Kanishka Dilshan
4.      *Bolg     : http://javaxclass.blogspot.com
5.      *Purpose   : Demonstrate howto use above classes
6.      */
7.     public class OOPDemo {
8.         public static void main(String args[]){
9.             //create 2 Point objects (p1 and p2)
10.            Point pnt1=new Point(3,4);
11.            Point pnt2=new Point(-2,7);
12.            System.out.println("x position of p1 : " + pnt1.getX());
13.            System.out.println("y position of p1 : " + pnt1.getY());
14.            System.out.println("distance frm Origin : " + pnt1.distance());

```

```

15. System.out.println("-----");
16.
17. //creating a Line object;
18. Line ln1=new Line(pnt1,pnt2);
19. //show details of ln1
20. System.out.println("Details of the line object");
21. ln1.showDetails();
22. double lineLen=ln1.getLength();
23. System.out.println("length of line : " + lineLen);
24. }
25. }

```

Output :

```

x position of p1      : 3
y position of p1      : 4
distance frm Origin  : 5.0
-----
Details of the line object
x1 : 3 y1 : 4
x2 : -2 y2 : 7
-----
length of line : 5.830951894845301

```

අපි ඉහත වැඩසටහනේ main method එකේදී සිදුකර ඇත්තේ අප විසින් ලියූ Point class එක සහ Line class එක භාවිතා කර objects සෑදීමයි. pnt1 හා pnt2 ලෙසින් Point object 2ක් සාදා ඉන් pnt1 එකෙහි details console එකෙහි print කර එම pnt1 හා pnt2 objects arguments ලෙස භාවිතා කරමින් Line object එක සාදා තිබෙන බව ඔබට වැටහෙනවා ඇති.

ඉහත Point හා Line class 2 යොදාගනිමින් සාදා ඇති පහත ඇප්ලටය භාවිතා කරමින් එම objects වල හැසිරීම පරීක්ෂා කර බලන්න. එම Java Applet එකෙහි source code එකද මම පහත ඉදිරිපත් කර ඇති link එකෙන් download කරගන්න පුලුවන්.

ජාවා ඇප්ලට ගැන අපි ඉදිරි පාඩම් පෙලකින් දීර්ඝව සාකච්ඡා කරමු. මෙහිදී අප උගත් දෑ ආදර්ශනය කිරීමට පමණක් පහත ජාවා applet එක ලියා ඇති බව සලකන්න

ජාවා වැඩසටහනකට Command Line Arguments ලබාගන්නා ආකාරය.

Tuesday, May 25, 2010 by Kaniskha Dilshan

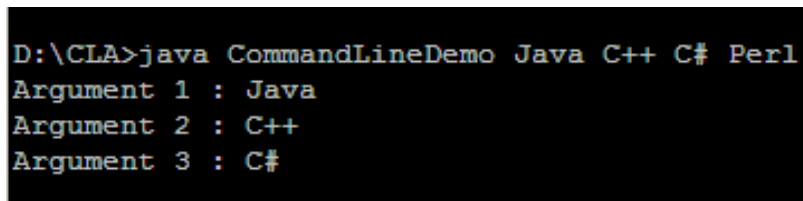
ජාවා වැඩසටහනක් ක්‍රියාත්මක කරවන(Execute) අවස්ථාවේම එයට යම් දත්තයක් හෝ කිහිපයක් ලබාදීමට Command Line Arguments භාවිතා කළ හැක. උදාහරණයක් ලෙස copy file1 file2 යන විධානයේ file1 හා file2 ලෙසට අපිවිසිත් දත්තයන් දෙකක් ලබාදී ඇත copy utility එක සඳහා එම file1 file2 command line argument 2කි. එම විධානය ක්‍රියාත්මක කළ විට copy utility එක විසින් අදාළ arguments 2 වැඩසටහන තුළට ලබාගෙන අදාළ කාණ්ඩය ඉටුකරනු ලබයි. අපි දැන් බලමු ජාවා වැඩසටහනකින් කොහොමද මෙවැනි හැකියාවක් ලබාදෙන්නේ කියලා.

අපි ජාවා වැඩසටහන ආරම්භයේදී ලබා දෙන arguments තුන්පත් වී තිබෙන්නේ **public static void main(String args[])** හි සඳහන් කරන ලද **args[]** නම් array එක තුළයි. එනම් main method එකෙන් parameter එකක් ලෙස ලබාදුන් array එක තුළයි. මෙය args ලෙසින්ම නම් වීම අත්‍යාවශ්‍ය නොවේ. නමුත් එය String type එකෙන් තිබීම අනිවාර්ය වේ. ඔබට arrays ගැන ගැටලුවක් ඇත්නම් [arrays සම්බන්ධ පාඩම](#) බලන්න.

example01:

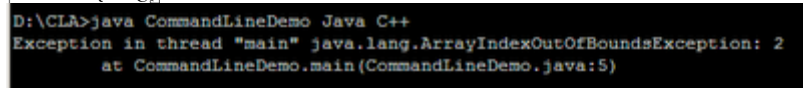
```
1. class CommandLineDemo{
2.     public static void main(String args[]){
3.         String argument1=args[0];
4.         String argument2=args[1];
5.         String argument3=args[2];
6.         System.out.println("Argument 1 : "+argument1);
7.         System.out.println("Argument 2 : "+argument2);
8.         System.out.println("Argument 3 : "+argument3);
9.     }
10. }
```

Output:



```
D:\CLA>java CommandLineDemo Java C++ C# Perl
Argument 1 : Java
Argument 2 : C++
Argument 3 : C#
```

අපි මෙහිදී Java C++ C# Perl ලෙසින් argument 4ක් දී තිබුනද console එකෙහි පෙන්වා ඇත්තේ Java C++ C# arguments 3 පමණි. අපි access කර ඇත්තේ **args[]** හි 0,1,2 යන elements 3 පමණි. යම්භෙයකින් අපි args හි අඩංගු නැති element එකක් access කළහොත් [ArrayIndexOutOfBoundsException](#) එකක් ලැබෙනු ඇත. එනම් program එකෙහි run time error එකක් ඇතිවනු ඇත. Error Control ගැන කෙරෙන පාඩමේදී අපි **Exception** ගැන විස්තරාත්මකව ඉගෙන ගනිමු. අපි ඉහත වැඩසටහනේ එවැනි error එකක් ලැබෙන අවස්ථාවක් ගැන බලමු.



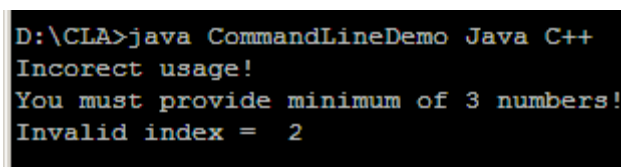
```
D:\CLA>java CommandLineDemo Java C++
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 2
at CommandLineDemo.main(CommandLineDemo.java:5)
```

අපි arguments ලෙස ලබා දී ඇත්තේ Java හා C++ පමණි නමුත් අප වැඩසටහන තුළදී args[] හි 3වන element එකද access කෙරේ(2nd index). එමනිසා මෙහිදී ඉහත සඳහන් කළ ආකාරයේ exception එකක් (මෙහිදී නම් run time error) ලැබේ.

අපි දැන් ඉහත ආකාරයේ දෝශ මගහැරවීමට සුදුසු පියවරක් යොදා එම වැඩසටහනම වැඩිදියුණු (modify) කරමු.

example02:

```
1. class CommandLineDemo{
2.     public static void main(String args[]){
3.         try{
4.             String argument1=args[0];
5.             String argument2=args[1];
6.             String argument3=args[2];
7.             System.out.println("Argument 1 : "+argument1);
8.             System.out.println("Argument 2 : "+argument2);
9.             System.out.println("Argument 3 : "+argument3);
10.        }catch(ArrayIndexOutOfBoundsException ex){
11.            System.out.println("Incorrect usage!");
12.            System.out.println("You must provide minimum of 3 numbers!");
13.            System.out.println("Invalid index = " + ex.getMessage());
14.        }
15.    }
16. }
17. }
```



```
D:\CLA>java CommandLineDemo Java C++
Incorrect usage!
You must provide minimum of 3 numbers!
Invalid index = 2
```


Exception handle කිරීම ගැන අපි ඉදිරි පාඩමකින් විස්තරාත්මකව සාකච්ඡා කරමු. මෙම පාඩමේ මූලික අරමුණ Command Line Arguments භාවිතා කරන ආකාරය හැදෑරීමයි. මෙහිදී මතක තබාගත යුතු කරුණාව වන්නේ args[] array එක String type එකෙන් ඇති බවයි එමනිසා අපි ගණිතමය දත්තයන් arguments ලෙස ගන්නා විට ඒවා යුද්ධ types වලට convert කිරීමට සිදුවනවා.

අපි දැන් ප්‍රායෝගික උදාහරණ කිහිපයක් බලමු.
Example03:

```
1. class CLADemo1{
2.     public static void main(String args[]){
3.         try{
4.             String name=args[0];
5.             float price=Float.parseFloat(args[1]);
6.             float VAT=Float.parseFloat(args[2]);
7.             float Total=price+VAT;
8.             System.out.println(name+" : " + Total);
9.         }catch(ArrayIndexOutOfBoundsException e){
10.            System.out.println("Incorrect usage!");
11.        }catch(NumberFormatException e){
12.            System.out.println("Invalid number "+ e.getMessage());
13.        }
14.    }
15. }
```

Output 01:

```
D:\CLA>java CLADemo1 Keyboard 800.0 75.50
Keyboard : 875.5
```

මෙහිදී numbers convert කරන විට ලබාදී ඇති number format එක දෝශ සහගත නම් එයද handle කිරීමට [NumberFormatException](#) එකක්ද program එක තුළදී catch කර ඇත. පහත රූපයෙන් එවැනි exception එකක් මතු වූ විට අප විසින් පෙන්වන දෝශ පණිවුඩය දැක්වේ.

```
D:\CLA>java CLADemo1 Keyboard 800.0 75.R0
Invalid number For input string: "75.R0"
```

ජාවා වස්තු පාදක ක්‍රමලේඛනය V (Data Encapsulation/ Data Hiding)

Thursday, June 3, 2010 by Kaniskha Dilshan



data සහ methods තනි ඒකකයකට සංක්ෂිප්ත කිරීම data encapsulation ලෙස හඳුන්වන්න පුළුවන්. මෙහි තනි ඒකකය ලෙස දක්වා ඇත්තේ ජාවා class එකකි. ජාවා class එකක ඉතාම සිත් ගන්නා සුලු ලක්ෂණයක් ලෙසටද data encapsulation දැක්විය හැකියි. මෙහිදී සිදුකරන්නේ පිටතින් දත්ත සඳහා සෘජුව පිවිසීමට(access) ඉඩ නොදී methods භාවිතා කොට ඒ සඳහා අතුරු මුහුණතක් සැකසීමයි. data hiding ලෙස හැඳින්වෙන්නේද මෙම සංකල්පයමයි. මෙම සංකල්පය ක්‍රියාවට නංවන ලද object එකක් අපට black box එකක් ලෙස හැඳින්විය හැකි අතර මෙහිදී එහි අභ්‍යන්තර ක්‍රියාවලිය ගැන සවිස්තර අවබෝධයක් නැතිව එහි සපයා ඇති methods ගැන පමණක් අවබෝධයක් ලබාගැනීමෙන් පසු අදාළ කාර්ය කිරීමට හැකිය.

Data Encapsulation සංකල්පය ක්‍රියාවට නැංවීම සඳහා අනුගමනය කළයුතු නීතිරීති.

- 1) Encapsulate කරන ලද object එකක අභ්‍යන්තර දත්ත(internal data) කිසි විටෙකත් වෙනත් පරිභාහිර object එකක් මගින් සෘජුව මෙහෙයවීම වැළැක්විය යුතුය.
- 2) දත්ත මෙහෙයවීම(manipulate) සඳහා accessor methods හා mutator methods භාවිතා කළ යුතුය.

- **Accessor Methods** - අභ්‍යන්තර දත්ත වලට අදාළ අගයන් පිටතට ලබාදීමට යොදාගනී. එමෙන්ම අභ්‍යන්තර දත්ත(internal data) ප්‍රදර්ශනය කිරීමක් වැනි ක්‍රියාවලටද යොදාගනී. සාමාන්‍යයෙන් accessor method එකක නම ආරම්භ වනුයේ get යන උපසර්ගයෙනි(prefix).

උදා : `getUserName(); , showErr();`

- **Mutator Methods** - අභ්‍යන්තර දත්ත සඳහා අගයන් සෙව කිරීම මෙම methods භාවිතයෙන් සිදුකෙරේ. මෙහිදී නව දත්ත වල වලංගු භාවය(validity) සහ සම්පූර්ණත්වය (integrity) සහතික කළ හැකිය. සාමාන්‍යයෙන් mutator method එකක නම ආරම්භ වනුයේ set යන උපසර්ගයෙනි(prefix).

උදා : `setWallpaper(String imagePath);`

දැන් අපි data encapsulation/data hiding සංකල්පය හොඳින් අවබෝධ කරගැනීම සඳහා උදාහරණයක් බලමු.

Cylinder.java

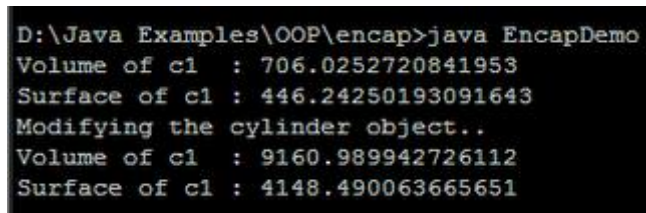
```
1.  /**
2.   *Author : Kanishka Dilshan
3.   *Purpose: Demonstrate encapsulation in Java
4.   *Blog   : http://javaxclass.blogspot.com
5.   */
6.
7.  public class Cylinder{
8.      //radius and height are private members
9.
10.     private double radius,height;
11.
12.     public Cylinder(double rad,double ht){
13.         radius=rad;
14.         height=ht;
15.     }
16.
17.     //mutator methods
18.     public void setRadius(double rad){
19.         radius=rad;
20.     }
21.
22.     public void setHeight(double ht){
23.         height=ht;
24.     }
25.
26.     //accessor methods
27.     public double getVolume(){
28.         return (Math.PI*radius*radius*height);
29.     }
30.
31.     public double getSurface(){
32.         return (2*Math.PI*radius*(height+radius));
33.     }
34. }
```

දැන් අපි ඉහත Cylinder class එක භාවිතා කිරීමට සරළ වැඩසටහනක් ලියමු.

EncapDemo.java

```
1.  public class EncapDemo{
2.      public static void main(String args[]){
3.          Cylinder c1=new Cylinder(4.23,12.56);
4.          //invoking accessor methods
5.          double surf1=c1.getSurface();
6.          double vol1=c1.getVolume();
7.          System.out.println("Volume of c1 : " + vol1 );
8.          System.out.println("Surface of c1 : " + surf1 );
9.
10.         //invoking mutator methods
11.         System.out.println("Modifying the cylinder object..");
12.         c1.setRadius(23.11);
13.         c1.setHeight(5.46);
14.         double surf2=c1.getSurface();
15.         double vol2=c1.getVolume();
16.         System.out.println("Volume of c1 : " + vol2 );
17.         System.out.println("Surface of c1 : " + surf2 );
18.
19.     }
20. }
```

Output :



```
D:\Java Examples\OOP\encap>java EncapDemo
Volume of c1 : 706.0252720841953
Surface of c1 : 446.24250193091643
Modifying the cylinder object..
Volume of c1 : 9160.989942726112
Surface of c1 : 4148.490063665651
```

Cylinder class එකෙහි data members සියල්ලම private ලෙසද පිටතින් invoke කරන methods සියල්ලම public ලෙසටද යොදා ඇත. එනිසා මෙහිදී Cylinder object එකක internal data භාවිතයට access(ප්‍රවේශනය) කළ නොහැක. නවද අදාළ දත්ත මෙහෙයවීම සඳහා accessor සහ mutator methods යොදා ඇත. එමනිසා මෙම Cylinder class එක encapsulate කරන ලද ඒකකයකි.

ජාවා වස්තු පාදක ක්‍රමලේඛනය VI (Inheritance) :: 1 කොටස

Saturday, June 12, 2010 by Kaniskha Dilshan

Inheritance යනු ජාවාහි එන ප්‍රභල සංකල්පයක්. එමෙන්ම වස්තු පාදක වැඩසටහන් (OOP programs) ලිවීමේදී මෙය බොහෝ විට භාවිතා වනවා. C++ වැනි අනෙකුත් OOP සඳහා සහය දක්වන පරිගණක භාෂා වලත් මෙම සංකල්පය සඳහා සහය දැක්වුවද C++ හා Java අතර inheritance හිදී යම් වෙනස්කම් පවතිනවා.

දැමී ඇති බලලමු මොකක්ද මේ inheritance ක්‍රියාවලිය කියලා, Inheritance යනු යම් class එකකින් සාදන ලද වස්තූන්(objects) වෙනත් class එකකට අයත් objects වල ගුණාංගයන්ද අයත් කර ගැනීමේදී සිදුකරන ක්‍රියාවලියයි. එම නිසා මෙහිදී අදාල ලක්ෂණ අත් කරගත් class එක සහ ලක්ෂණ අත් කරගැනීමට බඳුන් වූ class එක සඳහා පොදු ලක්ෂණ එකක් හෝ කිහිපයක් පවතිනවා. අදාල ලක්ෂණ අත් කරගත් class එක **child** class එක නොහොත් **sub** class එක ලෙසත් ලක්ෂණ අත් කරගැනීමට බඳුන් වූ class එක **parent** class එක නොහොත් **super** class එක ලෙසත් හැඳින්වෙනවා. මෙම ක්‍රියාවලියේදී **extends** නම් ජාවා keyword එක භාවිතා වනවා. යම් class එකක් Inherit කිරීමෙන් සාදන ලද නව class එක එහි parent class එකෙහි attributes සහ behaviour යනාදිය උකහා ගන්නා නිසා මෙහිදී ඉබේම software re-usability නම් සංකල්පයද ක්‍රියාත්මක වනවා. නමුත් inherit කිරීමෙන් සාදාගන්නා ලද නව class එකෙහි එහි parent class එකේ නොමැති attributes සහ behaviours අන්තර්ගත වීමට පුලුවන්.

ජාවා තුල inheritance සංකල්පය ක්‍රියාවට නැංවීමේදී දැන ගතයුතු කරුණු.

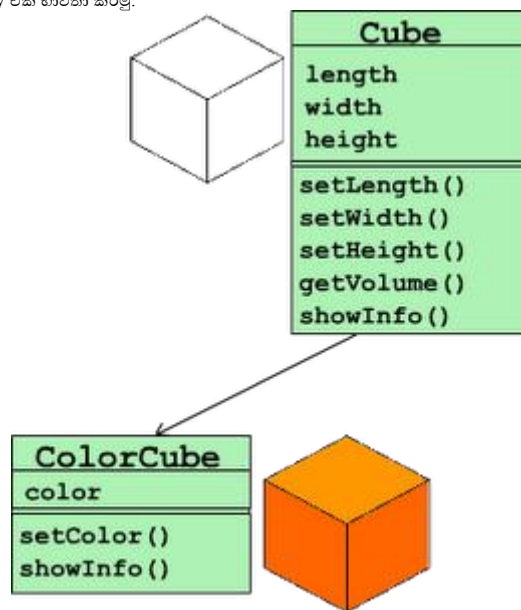
- ▶ අලුතින් සාදන සෑම class එකක්ම අනිවාර්යයෙන්ම වෙනත් class එකක් extend (inherit) කල යුතුය.
- ▶ extend කරන්නේ කුමන class එකක්ද යන්න විශේෂයෙන් සඳහන් කර නැති විට Object යන class එක inherit වීම සිදුවේ. Object යනු ජාවා class hierarchy එකේ උඩින්ම ඇති super class එකයි.
- ▶ inherit කිරීමකදී කිසිවිටෙකත් constructors උකහා ගැනීම සිදුනොවේ. සියලුම constructors එය define කරන ලද class එක සඳහා විශේෂ වේ. (constructor පිළිබඳ අපි මීට පෙර පාඩමකින් සාකච්ඡා කර තිබේ එම පාඩම සඳහා [මෙතන click කරන්න](#))
- ▶ යම්කිසි sub class එකක් construct වීමේදී එම class එකෙහි super class එකෙහි constructor එක මුලින්ම call වීම සිදුවේ.
- ▶ ජාවාහිදී ඕනෑම class එකකට තිබිය හැක්කේ එක් parent class එකක් පමණි. එනම් multiple inheritance සඳහා ජාවා සහය නොදක්වයි. එමනිසා multiple inheritance මගින් සිදුවන කාඩීය ජාවාහිදී සිදුකරන්නේ interfaces භාවිතයෙනි. අපි ඉදිරි පාඩමකින් ජාවා interfaces පිළිබඳ අධ්‍යයනය කරමු. ජාවා multiple inheritance සඳහා සහය නොදැක්වුවද C++ වලදී නම් multiple inheritance සඳහා ඉඩකඩ සලසා තිබේ.

Syntax for extending/inheriting a class

```
class SubClassName extends SuperClassName
```

දැන් අපි මෙම inheritance සංකල්පය හොඳින් වටහා ගැනීම සඳහා සරල ජාවා වැඩසටහනක් ලියමු.

එම ජාවා වැඩසටහන සඳහා අපි පහත hierarchy එක භාවිතා කරමු.



©<http://javaxclass.blogspot.com>

Cube.java

```
1.  /**
2.   *class : Cube
3.   *Author : Kanishka Dilshan
4.   *Purpose: Demonstrate inheritance in Java
5.   *Blog : http://javaxclass.blogspot.com
6.   */
7.
8.
9.  class Cube {
10.     protected int length;
11.     protected int width;
12.     protected int height;
13.
14.     public Cube(int l,int w,int h){
15.         length=l;
16.         width=w;
```

```

17.     height=h;
18.     }
19.
20.     public void setLength(int l){
21.         length=l;
22.     }
23.
24.     public void setWidth(int w){
25.         width=w;
26.     }
27.
28.     public void setHeight(int h){
29.         width=h;
30.     }
31.
32.     public int getVolume(){
33.         return(length*width*height);
34.     }
35.
36.     public void showInfo(){
37.         System.out.println("Length \t:"+this.length);
38.         System.out.println("Width \t:"+this.width);
39.         System.out.println("Height \t:"+this.height);
40.         System.out.println("Volume \t:"+getVolume());
41.     }
42.     }

```

ColorCube.java

```

1.     /**
2.      *class : ColorCube
3.      *Author : Kanishka Dilshan
4.      *Purpose: Demonstrate inheritance in Java
5.      *Blog : http://javaxclass.blogspot.com
6.      */
7.
8.     class ColorCube extends Cube {
9.         String color;
10.
11.         public ColorCube(int l,int w,int h,String color){
12.             //constructing the parent
13.             super(l,w,h);
14.             this.color=color;
15.         }
16.
17.         public void setColor(String color){
18.             this.color=color;
19.         }
20.
21.         public void showInfo(){
22.             super.showInfo();
23.             System.out.println("Color \t:"+this.color);
24.         }
25.     }

```

InheritanceDemo.java

```

1.     /**
2.      *class : InheritanceDemo
3.      *Author : Kanishka Dilshan
4.      *Purpose: Demonstrate inheritance in Java
5.      *Blog : http://javaxclass.blogspot.com
6.      */
7.
8.     class InheritanceDemo {
9.         public static void main(String args[]){
10.             Cube cube1=new Cube(3,7,5);
11.             ColorCube ccb1=new ColorCube(2,9,4,"Blue");
12.
13.             cube1.showInfo();
14.             System.out.println("_____");
15.             ccb1.showInfo();
16.
17.             System.out.println("\nMaking changes to cube objects....");
18.             cube1.setWidth(8);
19.             ccb1.setColor("Yellow");
20.             ccb1.setHeight(6); //this methods is inherited by the Cube class
21.             System.out.println("New info. of modified objects\n");
22.             cube1.showInfo();
23.             System.out.println("_____");
24.             ccb1.showInfo();
25.         }
26.     }

```

Output :

```
Length :3
Width :7
Height :5
Volume :105

Length :2
Width :9
Height :4
Volume :72
Color :Blue

Making changes to cube objects....
New info. of modified objects

Length :3
Width :8
Height :5
Volume :120

Length :2
Width :6
Height :4
Volume :48
Color :Yellow
```

ඉහත ඡායා වැඩසටහන අධ්‍යයනයෙන් ඔබට inheritance පිළිබඳ තවදුරටත් වටහාගත හැකිවනු ඇත. එහි class variables සඳහා **protected** access එකක් ලබාදීමෙන් සිදුකරන්නේ එම data members සඳහා ප්‍රවේශවීම(access) අදාළ class එක තුළදී සහ එහි sub classes වලට පමණක් සීමා කිරීමයි.

දැන් අපි Inheritance හිදී සිදුවන ක්‍රියාවලිය සියුම්ව වටහා ගැනීමට සුදුසු ඡායා වැඩසටහනක් අධ්‍යයනය කරමු.

InherianceStudy.java

```
1.  /**
2.   *class : InherianceStudy
3.   *Author : Kanishka Dilshan
4.   *Purpose: Demonstrate inheritance in Java
5.   *Blog : http://javaxclass.blogspot.com
6.   */
7.
8.   class Parent {
9.       protected int i,j;
10.
11.       public Parent(){
12.           System.out.println("Using the default constructor of Parent class..");
13.           i=0;
14.           j=0;
15.       }
16.
17.       public Parent(int i,int j){
18.           System.out.println("Using the integer constructor of Parent class..");
19.           this.i=i;
20.           this.j=j;
21.       }
22.
23.       public void showMessage(String msg){
24.           System.out.println("A message from Parent class..");
25.           System.out.println("Message : "+msg);
26.       }
27.   }
28.
29.   class Child extends Parent {
30.       protected String text;
31.
32.       public Child(){
33.           System.out.println("Using the default constructor of Child class..");
34.       }
35.
36.       public Child(int i,int j,String t){
37.           super(i,j);
38.           System.out.println("Using the integer constructor of Child class..");
39.           this.text=t;
40.       }
```

```

41.
42. public void showMessage(){
43.     showMessage("Java programming is very interesting!!");
44.     System.out.println("A message from Child class..");
45.     System.out.println("Message : "+text);
46. }
47. }
48.
49. class InheritanceStudy {
50.     public static void main(String args[]){
51.         System.out.println("Creating Parent objects...");
52.         System.out.println("_____\\n");
53.         Parent p1=new Parent();
54.         Parent p2=new Parent(2,5);
55.         System.out.println("\\nCreating Child objects...");
56.         System.out.println("_____\\n");
57.
58.         Child c1=new Child();
59.         Child c2=new Child(3,-1,"Java is owned by Sun Microsystems");
60.         System.out.println("\\nShowing messages..");
61.         System.out.println("_____\\n");
62.         c2.showMessage();
63.     }
64. }

```

Output:

```

Creating Parent objects...

_____

Using the default constructor of Parent class..
Using the integer constructor of Parent class..

Creating Child objects...

_____

Using the default constructor of Parent class..
Using the default constructor of Child class..
Using the integer constructor of Parent class..
Using the integer constructor of Child class..

Showing messages..

_____

A message from Parent class..
Message : Java programming is very interesting!!
A message from Child class..
Message : Java is owned by Sun Microsystems

```

ඉහත ඡායා කේතය දී ඇති ප්‍රතිඵලය සමග සැසඳීමෙන් ඔබට Inheritance සංකල්පය ගැන හොඳ අවබෝධයක් ලබා ගැනීමට පුලුවන්. Sub class එකකින් object එකක් සාදන සෑම විටම එහි super class එකක් construct වීම සිදුවන බව දැන් ඔබට වටහා ගත හැකිවිය යුතුයි.

ඡායා වැඩසටහනකින් Keyboard Inputs ලබාගන්නා අයුරු :: (Accessing the standard input stream)

Thursday, June 17, 2010 by Kaniskha Dilshan

ඡායා භාවිත යෙදුම්(Java Applications) Desktop applications හා Console applications වශයෙන් මූලික ආකාර 2ක් තිබෙනවා . Desktop applications සඳහා textbox වැනි component භාවිතාකර user inputs ලබාගන්නා පුලුවන් මේ සඳහා awt හා swing ලෙස ඡායා පැකේජ 2ක් හඳුන්වාදී තිබෙනවා. නමුත් console application යනු console එක(windows වලදී command prompt එක) ක්‍රියාකරන ආකාරය අනුව කොන්සෝල් එකෙහි inputs ලබාගැනීමට වන්නේ keyboard එක මගින් ලබාදෙන string එකක් වශයෙන්. අපි අද පාඩමෙන් අධ්‍යයනය කරන්නේ කොන්සෝල් එක මගින් user input එකක් ලබාගන්නා අයුරුයි.

keyboard එකකින් ලබාදෙන දත්ත ඡායා තුලදී stream එකක් ලෙස සලකනවා. ඒ විතරක් නොවෙයි file data සහ network connection යනාදියත් ඡායා තුලදී stream එකක් ලෙස සලකනවා. ඡායා තුල භාවිතාවන මූලිකම stream වර්ග 2 කමයි byte streams සහ character streams.

අපි දැන් සරළ ඡායා වැඩසටහනක් මගින් බලමු ඡායා වලදී stream යොදාගෙන keyboard input එකක් ලබාගන්නා ආකාරය.

KeybIn.java

```
import java.io.*;

public class KeybIn{
    public static void main(String args[]) throws IOException{
        String name;
        InputStreamReader myIsr=new InputStreamReader(System.in);
        BufferedReader myBr=new BufferedReader(myIsr);

        System.out.print("What is your name : " );
        name=myBr.readLine();

        System.out.println("Your name is : "+ name);
    }
}
```

දැන් අපි ඉහත වැඩසටහන අධ්‍යයනය කරමු.

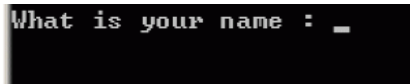
Line1 : මෙම වගන්තිය මගින් සිදුකරනුයේ IO නම් ජාවා පැකේජ එකේ තිබෙන ඕනෑම class එකක් හෝ සියල්ල අපගේ වැඩසටහන තුලදී භාවිතා විය හැකිය යන්න සඳහන් කිරීමයි.

Line4 : main method එකෙහි throws IOException යන්නෙන් විධාන කරන්නේ input හෝ output exception එකක් පැමිණියහොත් එය (දෝශයක් යයි කියමු) නොසලකා හරින්න(ignore) යන්නයි.

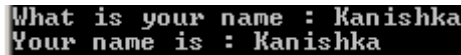
Line6 : මෙහිදී සිදුකරන්නේ System.in නොහොත් standard input stream එකෙන් දත්ත bytes වශයෙන් කියවා characters බවට පත්කලහැකි object එකක් සෑදීමයි. සාමාන්‍යයෙන් standard input stream එක යනු keyboard input stream එකයි.

Line7 : මෙහිදී සිදුකරන්නේ ඉහත සාදන ලද InputStremaReader object එක මගින් එකතුකරන ලද characters එකතු කර තනි ජේලියක් සෑදිය හැකි BufferedReader object එකක් සෑදීමයි.

Line10 : මෙහිදී කොන්සෝල් එකෙන් string එකක් read කිරීම සිදුකරනවා.



ඉහත ආකාරයේ ව්‍යුහයක් යොදාගනිමින් ජාවා තුලදී key board input එකක් ලබාගන්න පුළුවන්. ප්‍රතිඵලය :



ජාවාහි එන Wrapper classes යනු මොනවාද..?

Friday, June 18, 2010 by Kaniskha Dilshan

ඔබ දන්නවා ජාවා වැඩසටහන තිබෙනවා [මූලික දත්ත ආකාර](#) (primitive data types) 8ක්. ජාවාහි සෑම තැනකටම පොදු ඔබ්ජෙක්ට් ඔරියන්ටඩ් ගතිය මෙම primitive data types තුල තැනි බව බැලූ බැල්මටම පෙනෙනවා. මෙම wrapper classes නිර්මාණය කර ඇත්තේ මෙම primitive data type 8 සඳහා objects සෑදීමේ හැකියාව ලබාදීමටයි. එම නිසා ජාවා 1.0 සංස්කරණයේ සිටම මෙම primitive data types 8 සඳහා wrapper class 8ක් පැමිණෙනවා මෙවා java.lang පැකේජයේ අන්තර්ගත වන නිසා වැඩසටහනකට විශේෂයෙන් import කිරීමට අවශ්‍ය නැහැ.

මෙම wrapper classes යොදාගෙන ජාවාහි ඉතා වැදගත් කාඩ්භාරයක් කරගැනීමට පුළුවන්. data types convert කිරීම මෙවායේ සුලභ යෙදීමක්. ඕට් අමතරව එම දත්ත ආකාරය ගැන වැඩි විස්තර මෙම wrapper classes යොදාගනිමින් අපට ලබාගන්නටද පුළුවන්.

Primitive type	Wrapper class
boolean	Boolean
char	Character
byte	Byte
short	Short

int	Integer
long	Long
float	Float
double	Double

අපි දැන් බලමු wrapper class යොදාගෙන ලියන ලද සරල වැඩසටහනක්.

```

1. public class Wrapper{
2.     public static void main(String args[]){
3.         Integer num1=new Integer(23);
4.         Integer num2=new Integer(34);
5.         Integer tot1=new Integer(num1+num2);
6.         Integer tot2=num1+num2;
7.
8.         System.out.println("tot1 : " + tot2);
9.         System.out.println("tot2 : " + tot1);
10.    }
11. }
```

ඉහත වැඩසටහන නිරීක්ෂණයෙන් ඔබට පෙනෙනවා ඇති ක්‍රම 2කට අපි num1 හා num2 හි එකතුව ලබාගෙන තිබෙනවා. ජාවා 1.5 න් පසු සංස්කරණ වලදී නම් මෙම 2 ක්‍රම 2ම දෝශ රහිතයි නමුත් ඊට පෙර සංස්කරණයන්හි 2වන ක්‍රමය එනම් 6 වන පේලියේ ඇති ක්‍රමය දෝශ සහගතයි. Integer class type එක ඇති ඔබ්ජෙක්ට් එකකට int primitive type එකක් assign කිරීම ඇත්තටම වරදක්. නමුත් ජාවා 1.5න් පසු සංස්කරණ වලදී එලෙස assign කර ඇති අගය සුදුසු wrapper class type එකට convert කිරීමෙන් අනතුරුවයි assign කිරීම සිදුකරන්නේ. මෙය autoboxing ලෙසත් හඳුන්වනවා.

දැන් අපි බලමු මෙම wrapper classes භාවිතාකර data types convert කරන ආකාරය. මේ සඳහා ලියන ලද සරල ජාවා වැඩසටහන බලමු.

```

1. public class Wrapper{
2.     public static void main(String args[]){
3.         String str1="23.44";
4.         String str2="44.76";
5.         float val1=Float.parseFloat(str1);
6.         float val2=Float.parseFloat(str2);
7.
8.         System.out.println("str1 \t=> " + str1);
9.         System.out.println("str2 \t=> " + str2);
10.        System.out.println("String Data\t=> "+str1+str2);
11.        System.out.print("float  Data\t=> ");
12.        System.out.println(val1+val2);
13.    }
14. }
```

```

str1      => 23.44
str2      => 44.76
String Data => 23.4444.76
float  Data => 68.2
```

line 5 හා line6 දී සිදුකර ඇත්තේ str1 හා str2 යන String data val1 හා val2 ලෙස float බෙටා බවට පත්කිරීමයි. ඒ ආකාරයට wrapper class අටෙම ඇති parseXXX methods (parseInt (), parseFloat (), parseDouble ()...) දත්ත භාවිතා කර අපිට අවශ්‍ය data type වලට convert කරගන්න පුලුවන්.

දැන් අපි wrapper classes භාවිතා කර යම් data type එකකට දැරිය හැකි උපරිම හා අවම අගයන් ලබාගැනීමට වැඩසටහනක් ලියමු.

```

1. public class Wrapper{
2.     public static void main(String args[]){
3.
4.         System.out.println("Byte wrapper"+"(t_____");
5.         System.out.println("Max value : "+Byte.MAX_VALUE);
6.         System.out.println("Min value : "+Byte.MIN_VALUE);
7.         System.out.println();
8.
9.         System.out.println("Integer wrapper"+"(t_____");
10.        System.out.println("Max value : "+Integer.MAX_VALUE);
11.        System.out.println("Min value : "+Integer.MIN_VALUE);
12.        System.out.println();
13.
14.        System.out.println("Long wrapper"+"(t_____");
15.        System.out.println("Max value : "+Long.MAX_VALUE);
16.        System.out.println("Min value : "+Long.MIN_VALUE);
17.        System.out.println();
18.
19.        System.out.println("Float wrapper"+"(t_____");
20.        System.out.println("Max value : "+Float.MAX_VALUE);
21.        System.out.println("Min value : "+Float.MIN_VALUE);
```



```

22. System.out.println();
23.
24. System.out.println("Double wrapper"+"t_____");
25. System.out.println("Max value : "+Double.MAX_VALUE);
26. System.out.println("Min value : "+Double.MIN_VALUE);
27. System.out.println();
28. }
29. }

```

ප්‍රතිඵලය:

```

Byte wrapper
Max value : 127
Min value : -128

Integer wrapper
Max value : 2147483647
Min value : -2147483648

Long wrapper
Max value : 9223372036854775807
Min value : -9223372036854775808

Float wrapper
Max value : 3.4028235E38
Min value : 1.4E-45

Double wrapper
Max value : 1.7976931348623157E308
Min value : 4.9E-324

```

ඡායා static members කියන්නේ මොනවාද? මොකටද යොදාගන්නේ?

Wednesday, June 23, 2010 by Kanishka Dilshan

Static කියන්නේ ස්ථිතික කියන එකනේ සරලව කිව්වොත් පැතිරියන ස්වභාවය නැති කියන එක ඉතින් මේ ස්ථිතික ස්වභාවය ඡායා වැඩසටහන් වලට ලබාදීමට තමයි මෙම static ලක්ෂණය ඡායා වලට හඳුන්වාදී තිබෙන්නේ, මේ සඳහා **static** නම් keyword එක භාවිතා වනවා. සාමාන්‍යයෙන් static modifier එක යොදා ඇති fields බැඳී තිබෙන්නේ class එක සමගයි. static members වර්ග දෙකක් තිබෙනවා ඒ තමයි

1. Class variables
2. Class methods

අපි දැන් class variables ගැන බලමු. ඔබ දන්නවා සාමාන්‍ය instance variable නම් object එකකට එක බැගින් තිබෙනවා. උදාහරණයක් ලෙස Student class එකේ studentName එක instance variable එකක්. එනිසා එය Student object එකකට අනන්‍ය(unique) ලෙස පවතිනවා. සරලව කිව්වොත් සෑම object එකකටම studentName කියලා වෙන වෙනම instance පවතිනවා. නමුත් කිසියම් static variable එකක් තැන්පත් class variable එකක් Student class එකේ තියනවා නම් සෑම object එකකටම එක බැගින් static members නෑ. එකම memory location එකක ඇති variable එකක් ලෙසටම පවතින්නේ. සෑම object එකක්ම එය share කරගැනීම සිදුවෙනවා. එනම් object 1000ක් තිබුනොත් instance variable නම් 1000ක් පවතිනවා නමුත් static variable නම් පවතින්නේ එකම එකයි. පහත ඉදිරිපත්කර ඇති උදාහරණය අධ්‍යයනය කිරීමෙන් ඔබට මේ ගැන අවබෝධ කරගැනීමට පහසු වේ.

උදාහරණයට කලින් static methods තැනහොත් class methods ගැනත් බලලම ඉඳිමු. static methods පොදුවේ භාවිතා වන්නේ class එකෙන් objects සෑදීමෙන් තොරව අදාල method එක භාවිතා කිරීමටයි. class එකක් තුල ඇති static variable හි අගයන් පිටතට ගැනීමට(access කිරීමට) මෙම static methods භාවිතා කළයුතු වනවා. මෙම static methods invoke(call) කිරීම සිදුවන්නේ class එකේ නම් භරයයි.

දැන් අපි static සංකල්පය යොදාගැනීමට භාවිතා කරන syntax ගැන අධ්‍යයනය කරමු.

Syntax 1 : Declaring a static variable

[access modifier] static [datatype] [variable_name] ;

ex: **private static int studID;**

Syntax 2 : Declaring a static method

[access modifier] static [datatype] [method_name] ()

ex: **public static int getSum()**

දැන් අපි උදාහරණය බලමු

1. /**
2. *class : StaticDemo
3. *Author : Kanishka Dilshan
4. *Purpose: introduce static keyword in Java
5. *Blog : http://javaclass.blogspot.com
6. */
- 7.

```

8.  class Student {
9.      private String name;
10.     private int studId;
11.     private static int numberOfStudents;
12.
13.     public Student(String name){
14.         this.name=name;
15.         //generate a new ID and increment the total value by 1
16.         studId=1000+(numberOfStudents++);
17.     }
18.
19.     public static int getNumberOfStuds(){
20.         return numberOfStudents;
21.     }
22.
23.     public void showStudDetails(){
24.         System.out.println("Name : " + name);
25.         System.out.println("stID : " + studId);
26.     }
27. }
28.
29.
30. public class StaticDemo{
31.     public static void main(String args[]){
32.         Student std1=new Student("Sajith");
33.         Student std2=new Student("Kanishka");
34.         Student std3=new Student("Ravindu");
35.         Student std4=new Student("Sathiranga");
36.
37.         std1.showStudDetails();
38.         std2.showStudDetails();
39.         std3.showStudDetails();
40.         std4.showStudDetails();
41.         System.out.println("Total students : " + Student.getNumberOfStuds());
42.     }
43. }
44. }

```

ප්‍රතිඵලය:

```

Name : Sajith
stID : 1000
Name : Kanishka
stID : 1001
Name : Ravindu
stID : 1002
Name : Sathiranga
stID : 1003
Total students : 4

```

පැහැදිලි කිරීම:

මෙහි name හා studId යන ඒවා instance variable වේ. numberOfStudents යන්න static variable එකකි. student ලා කියක් ඉන්නවද කියන තමයි numberOfStudents එකේ තබාගන්නේ එය object එකින් එකට වෙනස් වන අගයක් නොවේ. එනිසයි අපි numberOfStudents යන්න static ලෙස යොදාගෙන තිබෙන්නේ. තවද මෙහි showStudDetails() යන method එකත් static මොකද එයත් object එකින් එකට වෙනස් වන්නේ නැහැ අනික එම method එක numberOfStudents නම් static variable එකත් access කරනවා. එම නිසා එය static විය යුතුයි. එමනිසා class එකේ නමයොදාගෙන call කිරීම කලහැකියි 41 වන පේළියේදී ඔබට මෙය දැකගත හැකියි. තවත් OOP සංකල්ප අපි ඉදිරි පාඩමකදී සාකච්ඡා කරමු.

ජාවා තුළ Abstract Classes යොදාගන්නා අයුරු

Saturday, July 10, 2010 by Kaniskha Dilshan

වස්තු පාදක(OO) ජාවා වැඩසටහන් ලිවීමේදී බහුල වශයෙන් යෙදෙන සංකල්පයක් තමයි abstraction කියන්නේ මේ සඳහා ජාවා පරිගණක භාෂාව තුළ පහසුකම් සපයා තිබෙනවා.

අපි මුලින්ම බලමු abstract class එකක ලක්ෂණ මොනවාද කියලා.

- Abstract class වලින් කිසිවිටෙකත් objects සෑදිය නොහැක.
- Abstract class එකක් extend කර(inherit කර) පසුව සාදනු ලබන class වලට හොඳ පදනමක් සැපයීම.

- Generic (පොදු) ස්වභාවයකින් යුතුවීම. එනම් abstract class එකෙහි child class වලට පොදු ලක්ෂණ එතුල කැටිවී තිබීම(උදාහරණය මගින් විස්තර කර ඇත.)
- Parent class එකෙහි Abstract methods ඇත්නම් එහි sub class නොහොත් child classes වලදී එම abstract methods අනිවාර්යයෙන්ම override කළ යුතුය(override කරනවා යන්නෙහි සරල අදහස නම් අදාළ abstract method එකෙහි body එක implement කළ යුතු බවයි)

දැන් අපි උදාහරණය බලමු

class : AbstractDemo.java

```

1.  /**
2.  *class : AbstractDemo
3.  *Author : Kanishka Dilshan
4.  *Purpose: Describe abstract concept in Java
5.  *Blog : http://javaclass.blogspot.com
6.  */
7.
8.  abstract class Shape{
9.  protected int color;
10. protected Point pos; //position
11.
12. public void setColor(int color){
13.     this.color=color;
14. }
15.
16. public int getColor(){
17.     return color;
18. }
19.
20. public void setPos(Point p){
21.     pos=p;
22. }
23.
24. public Point getPos(){
25.     return pos;
26. }
27.
28. abstract public float getArea();
29. /* This method is defined as abstract since
30.    it cannot be implemented since Shape is a
31.    common class(abstract class) */
32. }
33.
34. class Rectangle extends Shape {
35.     /*since this class inherits Shape class
36.     *the getArea() method must be overridden
37.     */
38.     //these are properties specific to a Rectangle
39.     private float width,height;
40.
41.     public Rectangle(float w,float h,Point p,int col){
42.         super.color=col;
43.         super.pos=p;
44.         this.width=w;
45.         this.height=h;
46.     }
47.
48.     public float getArea(){
49.         return (width*height);
50.     }
51. }
52.
53. class Circle extends Shape {
54.     /*since this class inherits Shape class
55.     *the getArea() method must be overridden
56.     */
57.     private float radius;
58.
59.     public Circle(float rad,Point p,int col){
60.         super.color=col;
61.         super.pos=p;
62.         this.radius=rad;
63.     }
64.
65.     public float getArea(){
66.         return (float)(Math.PI*Math.pow(radius,2));
67.     }
68. }
69.
70. class Point{
71.     //class definition for represent a Point
72.     protected int xPos;

```

```

73. protected int yPos;
74. public Point(int x,int y){
75.     xPos=x;
76.     yPos=y;
77. }
78.
79. public int getX(){
80.     return xPos;
81. }
82.
83. public int getY(){
84.     return yPos;
85. }
86. }
87.
88.
89. public class AbstractDemo{
90.     //main method
91.     public static void main(String args[]){
92.         Rectangle rect1=new Rectangle(12.0f,7.5f,new Point(10,-3),255);
93.         Circle c1=new Circle(4.3f,new Point(3,8),100);
94.
95.         //let's print object details for rect1
96.         println("rect1 belongs to   : "+ rect1.getClass().toString());
97.         println("colour val of rect1 : "+ rect1.getColor());
98.         println("x position of rect1 : "+ rect1.getPos().getX());
99.         println("y position of rect1 : "+ rect1.getPos().getY());
100.        println("area of rect1      : "+ rect1.getArea() );
101.
102.        println("-----");
103.        //let's print object details for c1
104.        println("c1 belongs to       : "+ c1.getClass().toString());
105.        println("colour val of c1    : "+ c1.getColor());
106.        println("x position of c1    : "+ c1.getPos().getX());
107.        println("y position of c1    : "+ c1.getPos().getY());
108.        println("area of c1         : "+ c1.getArea() );
109.    }
110.
111.    public static void println(Object o){
112.        System.out.println(o);
113.    }
114.
115. }

```

output :

```

rect1 belongs to      : class Rectangle
colour val of rect1  : 255
x position of rect1  : 10
y position of rect1  : -3
area of rect1        : 90.0
-----
c1 belongs to         : class Circle
colour val of c1      : 100
x position of c1      : 3
y position of c1      : 8
area of c1            : 58.088055

```

අපි දැන් ඉහත code එක විස්තරාත්මකව අධ්‍යයනය කරමු

අපි abstract class එක ලෙස මෙහිදී යොදාගෙන තිබෙන්නේ Shape නම් class එකයි. එයට හේතුව වනුයේ Circle සහ Rectangle යන class දෙකටම පොදු ලක්ෂණ Shape තුළ අන්තර්ගත වන නිසයි.

Line 28

```

1.    abstract public float getArea();

```

මෙහි getArea() නම් method එක abstract ලෙස සටහන් කර ඇත්තේ එය Shape හි sub classes වලට (specific) අනන්‍ය නිසා Shape class එක තුළ implement කළ නොහැකි නිසාය.

Line 48 - 50

```

1.    public float getArea(){
2.        return (width*height);
3.    }

```

මෙහිදී Rectangle class එකෙහි super class එක වන Shape class එකෙහි ඇති abstract method එක implement කර ඇත. super class එකෙහි abstract method යම් ප්‍රමාණයක් ඇත්නම් ඒ සියල්ලම එහි subclasses වලදී implement කිරීම අනිවාර්ය වේ.

මෙම පාඩමෙන් අප ලබාගත් දැණුම මිලින පාඩම වන interfaces පාඩමේදී අත්‍යාවශ්‍ය වන නිසා මෙය හොඳින් අධ්‍යයනය කරන්න.

ජාවා interface සංකල්පය සහ එහි යෙදීම්

Friday, July 16, 2010 by Kaniskha Dilshan

Interface එකක් කිව්වම එකපාරටම මතක් වෙන්නෙ GUI එකක් නැත්නම් web form එකක් නේ. නමුත් මෙම පාඩමේදී අපි සලකා බලන්නේ වස්තු පාදක සංකල්පයක් වන interface සංකල්පය පිළිබඳවයි. මෙම සංකල්පයත් ජාවා ක්‍රමලේඛනයේ යෙදීමේදී බහුල වශයෙන් භාවිතා වන්නක්. මෙම පාඩමේදී පෙර පාඩම වන "[ජාවා තුළ Abstract Classes යොදාගන්නා අයුරු](#)" යන පාඩමේදී උගත් කරුණු වැදගත් වන නිසා අවශ්‍ය නම් නැවත එය අධ්‍යයනය කරන්න.

interface එකක ඇති ප්‍රධාන ලක්ෂණ ගැන අපි සලකා බලමු.

- interface එකක් යනු සම්පූර්ණයෙන්ම abstract class එකකි.
- interface එකක ඇති methods සියල්ලම අර්ථදක්වා(define) ඇත්තේ method body එකකින් තොරවය(මෙයට හේතුව interface එක fully abstract වීමයි. එමනිසා මෙහි methods සියල්ලක්ම abstract වේ)
- interface එකක් තුළ class variables (උදා : `private int age;`) පැවතිය නොහැක. නමුත් final attributes පැවතිය හැක(උදා: `private final SIZE=512;`)

මීළඟට අපි සලකා බලමු ලියන ලද interface එකක් භාවිතා කිරීමේදී සැලකිය යුතු කරුණු.

- class එකකට අවශ්‍ය නම් interface එකකට වැඩි ප්‍රමාණයක් භාවිතා කළ හැක. එනම් class එකකට අවශ්‍ය නම් interface එකකට වැඩි ප්‍රමාණයක් implement කළ හැක. මේ සඳහා `implement` keyword එක භාවිතාවේ.
- යම් class එකක් interface එකක් implement කර ඇත්නම් එම interface එකෙහි ඇති සියලුම methods override කිරීම අනිවාර්යයෙන්ම කළ යුතුය.
- class එක abstract නම් එසේ අවශ්‍ය නොවේ.
- class වලදී මෙන් inheritance සංකල්පය interface සඳහාද භාවිතා කළ හැක. එනම් interface එකකදී වෙනත් interface එකක් extend කළ හැක.
- multiple inheritance නම් සංකල්පය C++ වලදී මෙන් java තුලදී කිරීමට ඉඩදී නොමැති වුවද, මෙම interface සංකල්පය යොදාගෙන එම ක්‍රියාවලිය ඉටුකරගත හැක.

අපි දැන් බලමු interface එකක් ලිවීම සඳහා භාවිතා වන syntax එක

```
1. interface MyInterface {
2.
3.     final int VAL=512;
4.
5.     void myMethod1();
6.     int myMethod2();
7.     .....
8.     .....
9. }
```

දැන් අපි බලමු interface සංකල්පයේ යෙදීම් ආදර්ශනය කිරීමට ලියන ලද සරල වැඩසටහනක්

```
1. /**
2.  *class : CarControllerDemo
3.  *Author : Kanishka Dilshan
4.  *Purpose: Describe interface concept in Java
5.  *Blog : http://javaclass.blogspot.com
6.  */
```

```

7.
8. interface CarController{
9.     final float MAX_SPEED=320f;
10.
11.     void accelerate(); //accelerate the car
12.     void applyBreak(); //apply breaks
13. }
14.
15. class BMW implements CarController{
16.     private String model;
17.     private float speed;
18.
19.     public BMW(String model){
20.         this.model=model;
21.         speed=0f;
22.     }
23.     public void accelerate(){
24.         //implementing accelerate() method in the interface
25.         if(speed < MAX_SPEED)
26.             speed+=15;
27.     }
28.
29.     public void applyBreak(){
30.         //implementing applyBreak() method in the interface
31.         if(!(speed-20) < 0))
32.             speed-=20;
33.         else
34.             speed=0;
35.     }
36.
37.     public void showDetails(){
38.         System.out.println("Car model : "+model);
39.         System.out.println("Speed    : "+speed);
40.     }
41. }
42.
43. //another car called Nissan implements CarController interface
44. class Nissan implements CarController{
45.     private String model;
46.     private float speed;
47.
48.     public Nissan(String model){
49.         this.model=model;
50.         speed=0f;
51.     }
52.     public void accelerate(){
53.         //implementing accelerate() method in the interface
54.         if(speed < MAX_SPEED)
55.             speed+=15;
56.     }
57.
58.     public void applyBreak(){
59.         //implementing applyBreak() method in the interface
60.         if(!(speed-25) < 0))
61.             speed-=25;
62.         else
63.             speed=0;
64.     }
65.
66.     public void showDetails(){
67.         System.out.println("Car model : "+model);
68.         System.out.println("Speed    : "+speed);
69.     }
70. }
71.
72.
73. public class CarControllerDemo{
74.     public static void main(String args[]){
75.         //creating BMW car object
76.         BMW car1=new BMW("BMW M6");
77.         Nissan car2=new Nissan("Altima Sedan");
78.
79.         //accelerate car1 twice
80.         car1.accelerate();
81.         car1.accelerate();
82.         //applyBreak() for car1
83.         car1.applyBreak();
84.         car1.showDetails();
85.
86.         //accelerate car2 4 times
87.         car2.accelerate();
88.         car2.accelerate();
89.         car2.accelerate();
90.         car2.accelerate();
91.         //applyBreak() for car2
92.         car2.applyBreak();

```

```
93.     car2.showDetails();
94.     }
95. }
```

Output:

```
Car model : BMW M6
Speed      : 10.0
Car model : Altima Sedan
Speed      : 35.0
```

පැහැදිලි කිරීම:

ඉහත උදාහරණයේ Line8 සිට Line13 දක්වා ඇත්තේ define කරන ලද interface එකයි. BMW සහ Nissan යන class තුළදී implement කර ඇත්තේ එම interface එක තුළ ඇති accelerate() සහ applybreak() යන methods දෙකයි. මීට පෙර පාඩම් වලදී උගත් සිද්ධාන්ත භාවිතා කරමින් ඉහත කේතයේ අනෙකුත් කොටස් අවබෝධ කිරීම ඔබගේ කාර්යයකි.

ජාවා ක්‍රමලේඛන භාෂාව යොදාගෙන වැඩසටහන් ලිවීමේදී පහත අවස්ථා වලදී මෙම interface භාවිතය ප්‍රායෝගිකව දැකගත හැකිය.

- Event handling
- Database connectivity

මෙම පාඩමත් සමග ජාවා වල මූලික OOP සංකල්ප කොටස අවසන් වේ. ඉදිරි පාඩම් වලදී ජාවාහි අතුරු මුහුණත් සැකසීම පිළිබඳ අධ්‍යයනය කරමු.

Java වැඩසටහන් සඳහා User Interfaces සැකසීම :: Swing

පැකේජය හැඳින්වීම I කොටස

Friday, July 23, 2010 by Kaniskha Dilshan

අතුරු මුහුණත්(Graphical User Interfaces) යනු යම් පරිගණක වැඩසටහනක් භාවිතා කිරීම පහසු කරවන්නක්. සාමාන්‍ය පරිගණක පරිශීලකයෙක් නම් මේ වකවානුවේ බොහෝ දුරට භාවිතා කරනුයේ මෙවැනි වැඩසටහන්. ජාවා පරිගණක භාෂාව යොදාගෙන මෙවැනි GUI සහිත වැඩසටහන් සැකසීමට විශේෂ පහසුකම් ලබාදී තිබෙනවා. මේ සඳහා යොදාගත හැකි ප්‍රධාන ජාවා පැකේජ 2ක තමයි

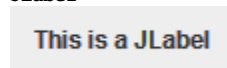
- AWT (Abstract Window Toolkit)
- Swing

විශේෂ හේතුවක් කිහිපයක් නිසා Swing පැකේජය භාවිතා කිරීම ප්‍රයෝජනවත් වනවා. ඉන් ප්‍රධානම හේතුව වන්නේ swing පැකේජය pure Java වලින් ලියැවුණු එකක් වීම. එම නිසා සුවිශේෂ වාසි කිහිපයක් අදාල වැඩසටහනට ලැබෙනවා. ඒට අමතරව AWT සහය දක්වන සියලුම පහසුකම් සඳහා swing ද සහය දක්වනවා. AWT හි නොමැති පහසුකම් ගතහොත් swing හි අඩංගු වනවා.

අපි මෙම පාඩම් මාලාවේදී වැඩිදුර සාකච්ඡා කිරීමට බලාපොරොත්තු වන්නේ Swing පැකේජය පිළිබඳවයි. NetBeans වැනි IDE එකක් යොදාගෙන ඉතා පහසුවෙන් GUI (අතුරු මුහුණත්) සාදාගත හැකිවුවද මෙය නවකයින්ගේ දැණුම මොට කරදරීම්මට එය හේතුවන බව මගේ අදහසයි. එම නිසා මුලදී Notepad හෝ Notepad++ වැනි text editor භාවිතා කර GUI සඳහා අදාල කේත ලිවීමෙන් ඒ සම්බන්ධ හොඳ අවබෝධයක් ලබාගෙන පසුව ක්‍රමලේඛන කටයුතු පහසු කරගැනීමට NetBeans භාවිතාකිරීම වරදක් නැහැ.

දැන් අපි බලමු swing components කිහිපයක විස්තර කිහිපයක්.

JLabel



JLabel එක වැදගත් වනුයේ user ට යම් විස්තරයක් පෙන්වීමේදීය.

JLabel එකක් සාදාගැනීම.

Constructors

```
JLabel(String text)
JLabel(Icon ico)
JLabel(String text, Icon ico, int align)
```

note : align සඳහා දියහැකි අගයන්

```
JLabel.LEFT
JLabel.CENTER
JLabel.RIGHT
JLabel.LEADING
```

ඉහත constructors යොදාගෙන JLabel 3ක් සාදන ආකාරය දැන් අපි බලමු.

Constructor 1

```
JLabel lbl1=new JLabel("Sri Lanka");
```

Constructor 2

```
ImageIcon ico=new ImageIcon("sriLanka.png");
JLabel lbl2=new JLabel(ico);
```

Constructor 3

```
JLabel lbl3=new JLabel("Sri Lanka", ico, JLabel.LEFT);
```

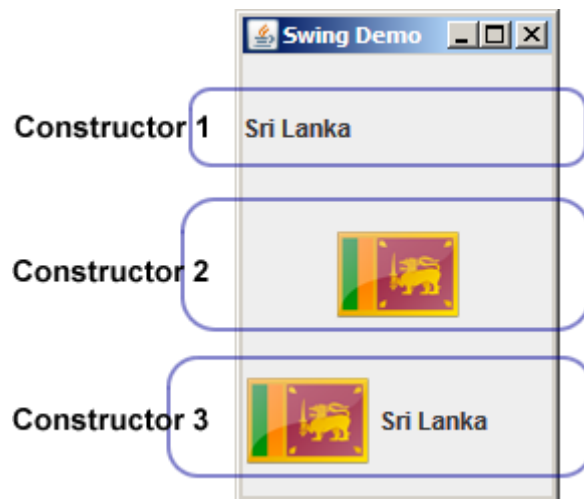
දැන් අපි බලමු ඉහත Label 3ක් යොදාගෙන සාදන ලද සරල GUI එකක් සඳහා කේතය.

Note: පහත කේතයේ ඇති සියලු කොටස් ඉදිරියේදී සාකච්ඡා කෙරේ එබැවින් එහි JLabel සම්බන්ධ කේත කොටස් පමණක් මෙහිදී අධ්‍යයනය කරන්න.

SwingDemo.java

```
1. import javax.swing.*;
2. import java.awt.*;
3.
4. class SwingDemo extends JFrame{
5.     public SwingDemo(){
6.         setTitle("Swing Demo");
7.         setLayout(new GridLayout(3,1));
8.         JLabel lbl1=new JLabel("Sri Lanka");
9.         ImageIcon ico=new ImageIcon("sriLanka.png");
10.        JLabel lbl2=new JLabel(ico);
11.        JLabel lbl3=new JLabel("Sri Lanka",ico,JLabel.LEFT);
12.        add(lbl1);
13.        add(lbl2);
14.        add(lbl3);
15.        setSize(300,300);
16.        setVisible(true);
17.    }
18.
19.    public static void main(String args[]){
20.        new SwingDemo();
21.    }
22. }
```

Output:



JLabel class එකෙහි methods

setText (String txt) -> JLabel
 getText () JLabel -> එකෙහි ඇති text එක ලබාගැනීම.

එකෙට text එකක් ලබා දීමට

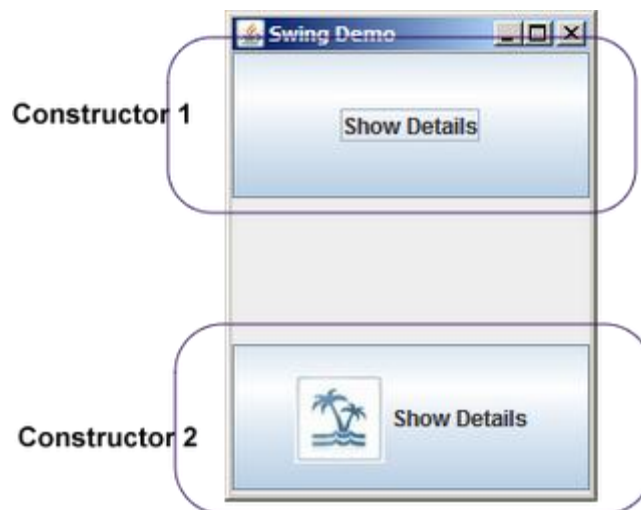
JButton

කිසියම් විධානයක් user ගේ අනුදැනුම මත ක්‍රියාත්මක කරවීම සඳහා යොදාගැනේ. මෙය Swing components අතුරින් බහුල වශයෙන් යෙදෙන component එකකි.

Constructors

JButton(String text)
 JButton(String text, Icon ico)

උදා:



Code:

1. ImageIcon ico=new ImageIcon("icon5.jpg");
2. JButton button2=new JButton("Show Details",ico);
3. add(button1);
4. JLabel lbl1=new JLabel("");
5. add(lbl1);
6. add(button2);

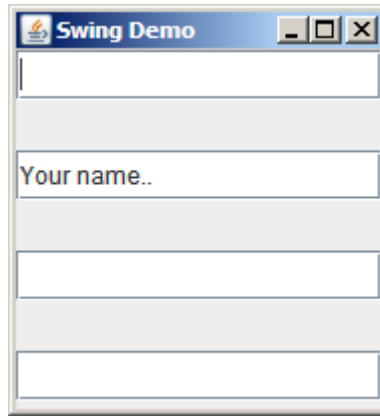
JTextField

User ගෙන් කිසියම් text input එකක් ලබාගැනීම සඳහා භාවිතා කෙරේ. (උදා: user name, age)

Constructors

JTextField()
 JTextField(String text)
 JTextField(int Columns)
 JTextField(String text,int Columns)

උදා:



Methods:

getText() user විසින් textbox එකට ඇතුළුකළ text එක ලබාගැනීමට
 setText(String str) textbox එකට text එකක් set කිරීම සඳහා

අපි ඊළඟ පාඩමෙන් අනෙකුත් swing components පිළිබඳ සාකච්ඡා කරමු.

Java වැඩසටහන් සඳහා User Interfaces සැකසීම :: Swing පැකේජය හැඳින්වීම II කොටස

Friday, July 23, 2010 by Kaniskha Dilshan

අපි ඉහත කලින් පාඩමේදී සාකච්ඡා කළේ JLabel, JButton හා JTextField පිළිබඳවයි. මෙම පාඩමෙන් තවත් swing components කිහිපයක් පිළිබඳව සාකච්ඡා කරමු.

JTextArea

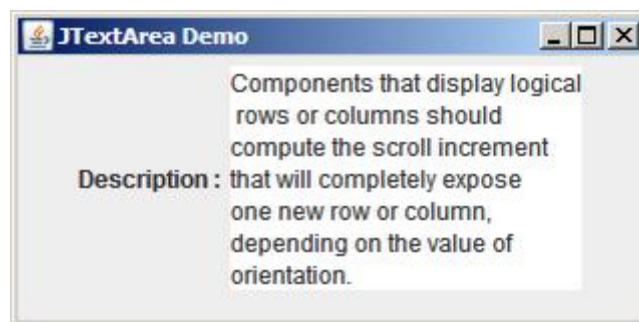
මෙය JTextField සඳහා බොහෝ සමාන වුවද මෙහි lines කිහිපයකින් යුතු user inputs ගත හැක. එනම් address එකක් හෝ කිසියම් description එකක් ගැනීමේදී මෙය භාවිතා කළ හැක.

JTextArea(int rows, int columns)

JTextArea(String text)

JTextArea(String text, int rows, int columns)

උදා:

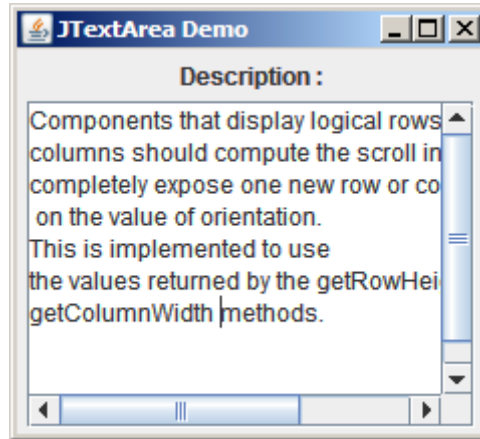


අපිට අවශ්‍ය නම් මෙම JTextArea සඳහා scrollbar යෙදිය හැක. මේ සඳහා කළ යුත්තේ JScrollPane එකක් තුළට JTextArea එක ඇමීමයි. මෙය සිදුකරන අයුරු පහත දැක්වේ.

```

1.  import javax.swing.*;
2.  import java.awt.*;
3.  class JTextAreaDemo extends JFrame{
4.
5.      public JTextAreaDemo(){
6.          setTitle("JTextArea Demo");
7.          JTextArea jtxt=new JTextArea(10,20);
8.          JScrollPane scrollpane=new JScrollPane(jtxt);
9.
10.         setLayout(new FlowLayout());
11.         setSize(300,300);
12.         setVisible(true);
13.         add(new JLabel("Description :"));
14.         add(scrollpane);
15.     }
16.
17.     public static void main(String args[]){
18.         new JTextAreaDemo();
19.     }
20. }
```

Output:

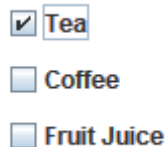


JScrollPane එකෙහි constructor එකට(line 8) JTextArea object එක වන jtxt object එක ලබාදීම මගින් scrollpane එකට textarea එක ඇතුළත් කිරීම සිදු කර ඇත.

JTextArea එකෙහි method ලෙස getText(),setText() බහුලව යෙදේ.

JCheckBox

චිකල්ප 1ක් හෝ කීපයක් තේරීමට ඉඩ ලබාදිය යුතුවීම මෙය භාවිතා කෙරේ.



isSelected() method එක භාවිතා කර යම් CheckBox එකක් select කර ඇත්දැයි පරීක්ෂා කල හැක. Event handling පාඩමේදී JCheckBox පිළිබඳ වැඩිදුරටත් සාකච්ඡා කෙරේ.

Java වැඩසටහන් සඳහා User Interfaces සැකසීම :: Swing

පැකේජය හැඳින්වීම III කොටස

Saturday, July 31, 2010 by Kaniskha Dilshan

JRadioButton

චිකල්ප කිහිපයක් දී ඇති විටදී ඉන් එක් චිකල්පයක් තෝරාගැනීමට අවකාශ සැලසීමට මෙම component එක යොදාගත හැක. උදාහරණයක් පහත ඉදිරිපත් කර ඇත. Mr,Mrs,Ms,Dr යන චිකල්ප වලින් එකක් පමණක් තෝරාගැනීමට මෙහිදී සිදුවේ. මෙම පාඩමෙන් සිදුකෙරෙන්නේ JRadioButton එකක් සාදාගන්නා ආකාරය පමණි. මෙවැනි event handling සිදුකිරීම ඉදිරි පාඩම් වලදී සාකච්ඡා කෙරේ.

JRadioButtonDemo.java

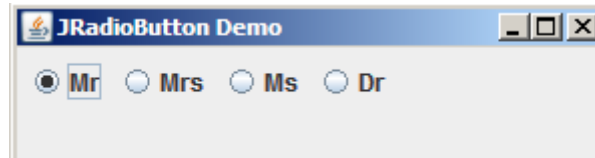
```
1. import javax.swing.*;
2. import java.awt.*;
3.
4. public class JRadioButtonDemo extends JFrame{
5.
6.     public JRadioButtonDemo(){
7.         setTitle("JRadioButton Demo");
8.         setLayout(new FlowLayout(FlowLayout.LEADING ));
9.         setSize(300,300);
10.        JRadioButton jrb_mr=new JRadioButton("Mr");
11.        JRadioButton jrb_mrs=new JRadioButton("Mrs");
12.        JRadioButton jrb_ms=new JRadioButton("Ms");
13.        JRadioButton jrb_dr=new JRadioButton("Dr");
14.        //grouping
15.        ButtonGroup btnGroup=new ButtonGroup();
16.        btnGroup.add(jrb_mr);
17.        btnGroup.add(jrb_mrs);
18.        btnGroup.add(jrb_ms);
19.        btnGroup.add(jrb_dr);
20.        //add them to the JFrame
21.        add(jrb_mr);
```

```

22.     add(jrb_mrs);
23.     add(jrb_ms);
24.     add(jrb_dr);
25.     setVisible(true);
26. }
27.
28. public static void main(String args[]){
29.     new JRadioButtonDemo();
30. }
31. }

```

Output:

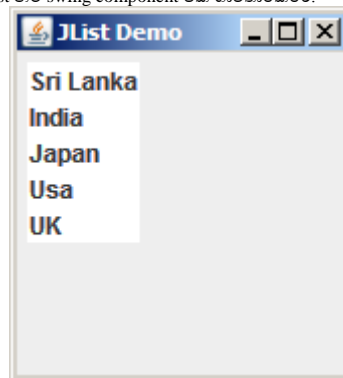


line14:

මෙහිදී අපි සාදාගත් JRadioButton ටික group කිරීම සිදුකෙරේ. මේවා group නොකළහොත් Mr, Mrs, Ms, Dr යන විකල්පයන්ගෙන් එකක් හෝ කිහිපයක් තේරීම සිදුකළ හැකිය. නමුත් JRadioButton යෙදීමේ අරමුණ මෙය නොවන නිසා සාමාන්‍යයෙන් JRadioButtons group කිරීම සිදුවේ.

JList

කිසියම් elements ප්‍රමාණයක් පහත ආකාරයට දැක්වීමට JList නම් swing component එක භාවිතාකෙරේ.



උදා: JListDemo.java

```

1.     import javax.swing.*;
2.     import java.awt.*;
3.
4.     public class JListDemo extends JFrame{
5.
6.     public JListDemo(){
7.         setTitle("JList Demo");
8.         setLayout(new FlowLayout(FlowLayout.LEADING ));
9.         setSize(300,300);
10.
11.         DefaultListModel dlm=new DefaultListModel();
12.         JList countries=new JList(dlm);
13.         dlm.addElement("Sri Lanka");
14.         dlm.addElement("India");
15.         dlm.addElement("Japan");
16.         dlm.addElement("Usa");
17.         dlm.addElement("UK");
18.
19.         add(countries);
20.         setVisible(true);
21.     }
22.
23.     public static void main(String args[]){
24.         new JListDemo();
25.     }
26. }

```

JList එක construct කරගැනීමට පෙර DefaultListModel එකක් සාදාගැනීම කළ යුතුයි. පසුව අපි සාදාගත් JList එකට දත්ත add කිරීමට JList එක clear කිරීමට, JList එකෙන් elements ඉවත් කිරීම මෙම DefaultListModel object එක හරහා සිදුවේ.Line11 දී මෙම DefaultListModel object එකක් සාදාගෙන ඇත.

Line12: මෙහිදී අපි සාදාගත් DefaultListModel object එක සහ JList එක සම්බන්ධ කිරීම සිදුකෙරේ.

දැන් අපි JList එක හා සම්බන්ධ වැදගත් method කිහිපයක් සලකා බලමු.

JList Methods	විස්තරය
<code>int getSelectedIndex()</code>	user විසින් list එකීනි මේ වනවිට select කර ඇති item එකෙහි index එක ලබාදේ.
<code>void setSelectedIndex()</code>	Program එක තුළින් list එකෙහි item එකක් select කර ඇති ලෙසට දක්වයි.
<code>Object getSelectedValue()</code>	user විසින් list එකීනි මේ වනවිට select කර ඇති item එක ලබාදේ.
<code>boolean isSelectedIndex(int idx)</code>	ලබාදෙන int value එකක් select වී ඇති item එකේ index value එක නම් true ලෙසටද නැතිනම් false ලෙසටද return කරයි.
<code>setModel(ListModel model)</code>	සකස් කරගත් ListModel object එකක් JList එක හා සම්බන්ධ කිරීමට යොදාගැනේ.
DefaultListModel Methods	විස්තරය
<code>void add(int index, Object element)</code>	ලබාදෙන index අගයකට(location එකකට) item එකක් ඇතුළු කිරීම සිදුකරයි.
<code>void addElement(Object obj)</code>	list එකේ අගට item එකක් ඇතුළු කිරීම සිදුකරයි.
<code>int capacity()</code>	දැනට list එකේ capacity එක නොහොත් items ගනණ ලබාදේ.
<code>void clear()</code>	list එකේ ඇති සියලු items clear කිරීම සිදුකරයි.
<code>boolean contains(Object obj)</code>	ලබාදෙන Object එකක් list එක තුළ අන්තර්ගත වේනම් true ලෙසද නැත්නම් false ලෙසටද ලබාදේ.
<code>Object elementAt(int index)</code>	ලබාදෙන index එකක ඇති value එක ලබාදේ.
<code>boolean isEmpty()</code>	list එක empty නම් true ලෙසටද නැත්නම් false ලෙසටද ලබාදේ.
<code>Object remove(int index)</code>	ලබාදෙන index එකක ඇති item එක ඉවත්කරයි.
<code>void removeAllElements()</code>	list එක තුළ ඇති සියලු items ඉවත් කරයි.

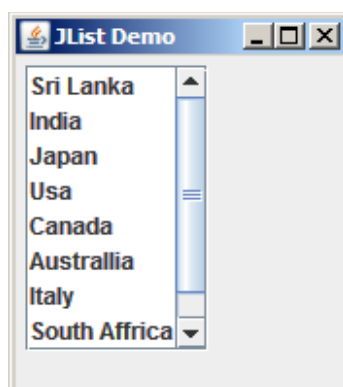
JList එකට scroll bar යෙදීමට අවශ්‍ය නම් JScrollPane එකක් යොදාගත හැකිය. එය සිදුකරන අයුරු පහත දැක්වේ.

```

1.  import javax.swing.*;
2.  import java.awt.*;
3.
4.  public class JListDemo extends JFrame{
5.
6.      public JListDemo(){
7.          setTitle("JList Demo");
8.          setLayout(new FlowLayout(FlowLayout.LEADING ));
9.          setSize(300,300);
10.
11.         DefaultListModel dlm=new DefaultListModel();
12.         JList countries=new JList(dlm);
13.
14.         JScrollPane scrollIP=new JScrollPane(countries);
15.         scrollIP.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);
16.
17.         dlm.addElement("Sri Lanka");
18.         dlm.addElement("India");
19.         dlm.addElement("Japan");
20.         dlm.addElement("Usa");
21.         dlm.addElement("Canada");
22.         dlm.addElement("Australia");
23.         dlm.addElement("Italy");
24.         dlm.addElement("South Affrica");
25.         dlm.addElement("Thailand");
26.
27.         add(scrollIP);
28.         setVisible(true);
29.     }
30.
31.     public static void main(String args[]){
32.         new JListDemo();
33.     }
34. }

```

Output:



GUI සැකසීම සඳහා Basic Layouts යොදාගන්නා අයුරු

Wednesday, August 11, 2010 by Kaniskha Dilshan

අපි මෙම පාඩමෙන් සාකච්ඡා කරන්නේ ජාවා වැඩසටහන් සඳහා අතුරුමුහුණත් සැකසීමේදී components සුදුසු පරිදි ස්ථානගත කිරීම සඳහා layouts යොදාගන්නා ආකාරයයි. මෙම layouts java.awt පැකේජයේ අන්තර්ගත වනවා. අප මෙම පාඩමේදී මූලික වශයෙන් පහත දැක්වෙන layouts භාවිතා කරන අයුරු සලකා බලමු.

- FlowLayout
- BorderLayout
- GridLayout

අපි මුලින්ම බලමු ජාවා අතුරුමුහුණතකට Layout manager එකක් set කරන්නේ කොහොමද කියලා.

syntax:

```
void setLayout(LayoutManager layoutObj);
```

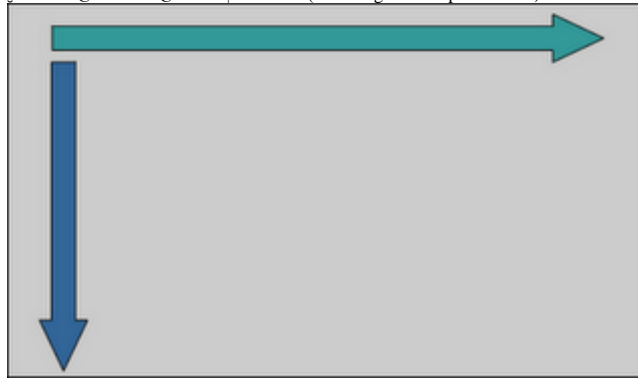
උදා:

1. this.setLayout(new FlowLayout());

දැන් අපි ඉහත සඳහන් කළ layouts විස්තරාත්මකව සලකා බලමු

FlowLayout

Panel class එකෙහි සහ එහි සියලුම child classes (උදා:Applet,JApplet,JPanel,...) වල default layout එක වන්නේ මෙයයි. සාමාන්‍යයෙන් components place(ස්ථානගත) කරනුයේ වමේ සිට දකුණට සහ උඩ සිට පහලට යන ආකාරයෙනි(left to right and top to bottom).



constructor:

1. FlowLayout(int align, int hgap, int vgap);

align: මෙමගින් සඳහන් කරනුයේ components ස්ථානගත කරනුයේ මොන පැත්තෙන් ආරම්භ වන පරිදිද යන්නයි

```
FlowLayout.LEFT  
FlowLayout.CENTER  
FlowLayout.RIGHT
```

hgap, vgap : මෙමගින් සඳහන් කරනුයේ components අතර horizontal gap එක සහ vertical gap එකයි මෙය pixels වලින් ලබාදිය යුතුය.

දැන් අපි FlowLayout එක යොදාගෙන සරල විත්තමුහුණතක් (GUI)සකසන අයුරු බලමු.



Code:

- ```
1. import java.awt.FlowLayout;
2. import javax.swing.*;
3.
4. public class FlowLayoutDemo extends JFrame{
5.
6. public FlowLayoutDemo(){
7. setTitle("FlowLayoutDemo Demo");
8. JPanel myPanel=new JPanel();
9. myPanel.setLayout(new FlowLayout(FlowLayout.CENTER,5,5));
10. //creating JButton objects
11. JButton btn1=new JButton("Component1");
12. JButton btn2=new JButton("Component2");
13. JButton btn3=new JButton("Component3");
14. JButton btn4=new JButton("Component4");
```

```

15. JButton btn5=new JButton("Component5");
16. JButton btn6=new JButton("Component6");
17. //adding JButton objects to the panel
18. myPanel.add(btn1);
19. myPanel.add(btn2);
20. myPanel.add(btn3);
21. myPanel.add(btn4);
22. myPanel.add(btn5);
23. myPanel.add(btn6);
24. this.add(myPanel);
25. this.setSize(492,100);
26. this.setDefaultCloseOperation(3);
27. this.setVisible(true);
28. }
29.
30. public static void main(String args[]){
31. new FlowLayoutDemo();
32. }
33. }

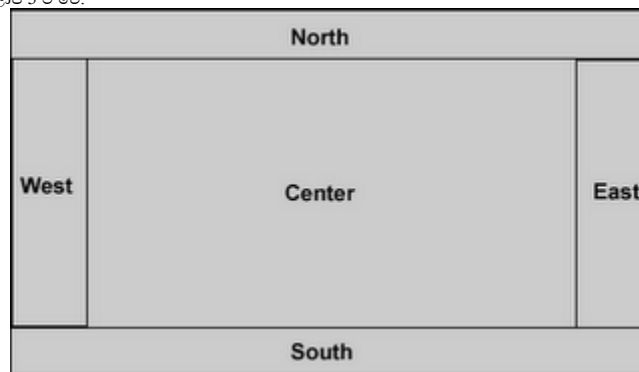
```

## BorderLayout

Window සහ එහි sub class සඳහා default layout එක වේ. මෙහිදී container එක කලාප 5 කට බෙදනු ලැබේ ඒවා නම්

1. North
2. South
3. East
4. West
5. Center

components place කරනුයේ මෙම කලාප 5 ට වේ.



Constructor:

1. BorderLayout(int hgap, int vgap)

**hgap, vgap** : මෙමගින් සඳහන් කරනුයේ components අතර horizontal gap එක සහ vertical gap එකයි මෙය pixels වලින් ලබාදිය යුතුය.

දැන් අප BorderLayout එක යොදාගෙන සරල චිත්‍රකමුහුණතක් (GUI) සකසන අයුරු බලමු.

Code:

```

1. import java.awt.BorderLayout;
2. import javax.swing.*;
3.
4. public class BorderLayoutDemo extends JFrame{
5.
6. public BorderLayoutDemo(){
7. setTitle("BorderLayout Demo");
8. JPanel myPanel=new JPanel();
9. myPanel.setLayout(new BorderLayout(5,5));
10. //creating JButton objects
11. JButton btn1=new JButton("North");
12. JButton btn2=new JButton("East");
13. JButton btn3=new JButton("South");
14. JButton btn4=new JButton("West");
15. JButton btn5=new JButton("Center");
16. //adding JButton objects to the panel
17. myPanel.add(btn1, BorderLayout.NORTH);
18. myPanel.add(btn2, BorderLayout.EAST);
19. myPanel.add(btn3, BorderLayout.SOUTH);
20. myPanel.add(btn4, BorderLayout.WEST);
21. myPanel.add(btn5, BorderLayout.CENTER);
22.
23. this.add(myPanel);

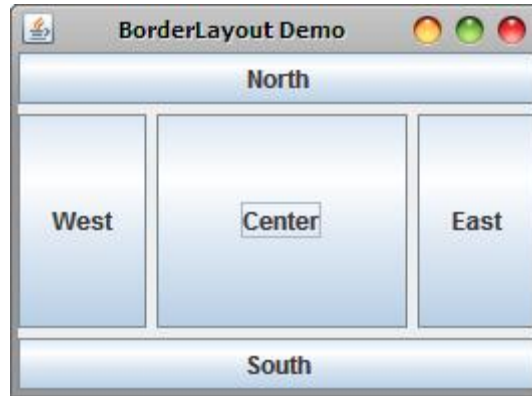
```

```

24. this.pack();
25. this.setDefaultCloseOperation(3);
26. this.setVisible(true);
27. }
28.
29. public static void main(String args[]){
30. new BorderLayoutDemo();
31. }
32. }

```

output:



### GridLayout

මෙම layout එක බොහෝ දුරට FlowLayout එකට සමානය නමුත් components place කරනුයේ සමාන cell sizes ඇති virtual grid එකක් තුළටය. මෙහිදී අදාළ component එකෙහි preferred size එක නොසලකා හැරේ.

Constructor:

1. GridLayout(int rows, int cols, int hgap, int vgap) ;

**rows, cols** : මෙමගින් දැක්වෙන්නේ components add කරනු ලබන virtual grid එකෙහි rows සහ columns ප්‍රමාණයයි.

**hgap, vgap** : මෙමගින් සඳහන් කරනුයේ components අතර horizontal gap එක සහ vertical gap එකයි මෙය pixels වලින් ලබාදිය යුතුය.  
Code:

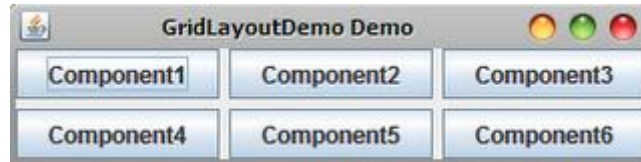
```

1. import java.awt.GridLayout;
2. import javax.swing.*;
3.
4. public class GridLayoutDemo extends JFrame{
5.
6. public GridLayoutDemo(){
7. setTitle("GridLayoutDemo Demo");
8. JPanel myPanel=new JPanel();
9. myPanel.setLayout(new GridLayout(2,3,5,5));
10. //creating JButton objects
11. JButton btn1=new JButton("Component1");
12. JButton btn2=new JButton("Component2");
13. JButton btn3=new JButton("Component3");
14. JButton btn4=new JButton("Component4");
15. JButton btn5=new JButton("Component5");
16. JButton btn6=new JButton("Component6");
17. //adding JButton objects to the panel
18. myPanel.add(btn1);
19. myPanel.add(btn2);
20. myPanel.add(btn3);
21. myPanel.add(btn4);
22. myPanel.add(btn5);
23. myPanel.add(btn6);
24.
25. this.add(myPanel);
26. this.pack();
27. this.setDefaultCloseOperation(3);
28. this.setVisible(true);
29. }
30.
31. public static void main(String args[]){
32. new GridLayoutDemo();
33. }
34. }

```

output:





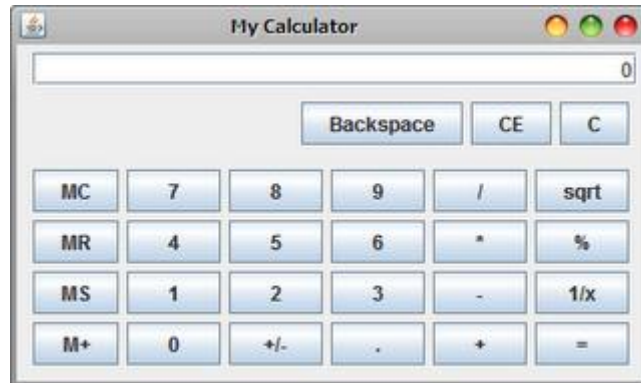
අපි ඉහත සාකච්ඡා කළ layout managers කිහිපය GUI සැදීමේදී බහුල වශයෙන් යෙදේ.

වඩාත් සංකීර්ණ GUI සැදීම

මෙහිදී සිදු කෙරෙනුයේ Layouts කිහිපයක් සංයුක්ත කොට කොට භාවිතා කිරීමය. Panel objects කිහිපක් සුදුසු පරිදි යෙදවීමෙන් අපට වඩාත් සංකීර්ණ GUI සකසාගත හැකිය.

දැන් අපි Windows calculator එකෙහි අනුරූපීය අප ඉගෙනගත් LayoutManagers යොදාගෙන සැදීමට උත්සාහ කරමු.

Output:



code:

```

1. import java.awt.*;
2. import javax.swing.*;
3.
4. public class ComplexGUIDemo extends JFrame{
5.
6. public ComplexGUIDemo(){
7. //this.setLayout(new GridLayout(3,1));
8. this.setLayout(new FlowLayout(FlowLayout.RIGHT));
9. //main panels
10. JPanel txtPanel=new JPanel();
11. JPanel numPanel1=new JPanel();
12. JPanel numPanel2=new JPanel();
13. //sub panels of numPanel2
14. JPanel numPanel2Sub1=new JPanel();
15. JPanel numPanel2Sub2=new JPanel();
16.
17. JTextField textBox=new JTextField("0",33);
18. textBox.setHorizontalAlignment(JTextField.RIGHT);
19. txtPanel.setLayout(new GridLayout(1,1));
20. txtPanel.add(textBox);
21.
22. numPanel1.setLayout(new FlowLayout(FlowLayout.RIGHT));
23. JButton btnbacksp=new JButton("Backspace");
24. JButton btncce=new JButton("CE");
25. JButton btnc=new JButton("C");
26. numPanel1.add(btnbacksp);
27. numPanel1.add(btncce);
28. numPanel1.add(btnc);
29.
30. numPanel2Sub1.setLayout(new GridLayout(4,1,5,5));
31. JButton btnMC=new JButton("MC");
32. JButton btnMR=new JButton("MR");
33. JButton btnMS=new JButton("MS");
34. JButton btnMPI=new JButton("M+");
35. numPanel2Sub1.add(btnMC);
36. numPanel2Sub1.add(btnMR);
37. numPanel2Sub1.add(btnMS);
38. numPanel2Sub1.add(btnMPI);
39.
40. numPanel2Sub2.setLayout(new GridLayout(4,5,5,5));
41. JButton btn7=new JButton("7");
42. JButton btn8=new JButton("8");
43. JButton btn9=new JButton("9");
44.
45. JButton btndv=new JButton("/");
46. JButton btnsqrt=new JButton("sqrt");
47. numPanel2Sub2.add(btn7);
48. numPanel2Sub2.add(btn8);
49. numPanel2Sub2.add(btn9);
50. numPanel2Sub2.add(btndv);

```

```

51. numPanel2Sub2.add(btnsqrt);
52.
53. JButton btn4=new JButton("4");
54. JButton btn5=new JButton("5");
55. JButton btn6=new JButton("6");
56.
57. JButton btnmul=new JButton("*");
58. JButton btnperc=new JButton("%");
59. numPanel2Sub2.add(btn4);
60. numPanel2Sub2.add(btn5);
61. numPanel2Sub2.add(btn6);
62. numPanel2Sub2.add(btnmul);
63. numPanel2Sub2.add(btnperc);
64.
65. JButton btn1=new JButton("1");
66. JButton btn2=new JButton("2");
67. JButton btn3=new JButton("3");
68.
69. JButton btnmin=new JButton("-");
70. JButton btn1bx=new JButton("1/x");
71. numPanel2Sub2.add(btn1);
72. numPanel2Sub2.add(btn2);
73. numPanel2Sub2.add(btn3);
74. numPanel2Sub2.add(btnmin);
75. numPanel2Sub2.add(btn1bx);
76.
77. JButton btn0=new JButton("0");
78. JButton btnPM=new JButton("+/-");
79. JButton btnpt=new JButton(".");
80.
81. JButton btnpl=new JButton("+");
82. JButton btneq=new JButton("=");
83. numPanel2Sub2.add(btn0);
84. numPanel2Sub2.add(btnPM);
85. numPanel2Sub2.add(btnpt);
86. numPanel2Sub2.add(btnpl);
87. numPanel2Sub2.add(btneq);
88.
89. numPanel2.setLayout(new FlowLayout());
90. numPanel2.add(numPanel2Sub1);
91. numPanel2.add(numPanel2Sub2);
92.
93. this.add(txtPanel);
94. this.add(numPanel1);
95. this.add(numPanel2);
96.
97. this.setTitle("My Calculator");
98. this.setDefaultCloseOperation(3);
99. this.setVisible(true);
100. this.setSize(390,235);
101. }
102.
103. public static void main(String args[]){
104. new ComplexGUIDemo();
105. }
106. }

```

# Errors සහ Exceptions කළමනාකරණය කිරීම (Exception Handling)

Monday, August 23, 2010 by Kaniskha Dilshan



කිසියම් වැඩසටහනක් 100% ක්ම දෝශ රහිතව ලිවීම ඉතාම අසීරු කාරණයකි. වැඩසටහනේ විශාලත්වය වැඩිවත්ම මෙහි අසීරුතාව සිසුයෙන් වැඩිවේ. Program එක සාර්ථකව compile වීම යනු එම වැඩසටහන දෝශරහිත යයි සැලකීමට සාධකයක් නොවේ. Program එකෙහි logical errors තිබිය හැක. logical errors යනු වැඩසටහන නියමාකාර ලෙස ක්‍රියානොකිරීමට(අපේක්ෂා කළ පරිදි ක්‍රියා නොකිරීමට) හේතුපාදක වන කාරණයකි.

ජාවා වැඩසටහනක ඇතිවිය හැකි errors ආකාර දෙකකි.

- Compile-time errors
- Run-time errors

සියලුම syntax errors හෙවත් භාෂාවේ කාරක රීති වලට පටහැනි අවස්ථා Compile-time errors ලෙස සැලකේ. මෙහිදී ජාවා compiler එක .java file එක මගින් .class file එක නොසාදයි. එනම් වැඩසටහන compile නොකෙරේ.

\*Compile-time errors සඳහා උදාහරණ.

```
1 class SomeError{
2 public static void main(String args[]){
3 System.out.println("Oops")
4 }
5 }
```

ඉහත වැඩසටහනේ 3 වන line එකෙහි ඇති statement එක semicolon ";" එකකින් අවසන් කර නැත එය syntax error එකකි. එමනිසා එය compile කිරීමේදී පහත දෝශය පෙන්වයි.

```
SomeError.java:3: ';' expected
System.out.println("Oops")
1 error
```

\* Run-time errors සඳහා උදාහරණ

මෙහිදී program එක සාර්ථකව compile වුවද එය run කිරීමට යාමේදී විවිධ ගැටලු ඇතිවේ.

- නිඛිල සංඛ්‍යා 0න් බෙදීම
- Array එකක සීමාවෙන් පිට element එකක් access කිරීමට යාම
- කිසියම් array එකක් තුළට නොගැලපෙන data type එකකින් යුතු අගයක් ඇතුළු කිරීම
- null object එකක attribute හෝ methods භාවිතා කිරීම(access)

මේ සඳහා තවත් උදාහරණ රාශියක් ඇත.

දැන් අපි Run-time errors සඳහා උදාහරණ කිහිපයක් බලමු.

\*Division by zero

```
1. class Runtime1{
2. public static void main(String args[]){
3. int a=5;
4. int b=6;
5. int c=10;
6. int res=c/(b-a-1);
7. System.out.println(res);
8. }
9. }
```

මෙම වැඩසටහන සාර්ථකව compile වුවද run වීමේදී පහත දෝශය පෙන්වයි. එයට හේතුව line 6 දී (b-a-1) කොටසෙහි අගය 0 වීමයි.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
at Runtime1.main(Runtime1.java:6)
```

\* Array index out of bounds

```
1. class Runtime2{
2. public static void main(String args[]){
3. int arr[]=new int[5];
4. int i=arr[10];
5. }
```

6. }

මෙය run වීමේදී පහත දෝශය පැන නගී

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
at Runtime2.main(Runtime2.java:4)
```

දැන් අපි Run-time errors පිළිබඳ පුළුල්ව සාකච්ඡා කරමු.  
Run-time errors වර්ග 2කි

1. Errors
2. Exceptions

අපි මුල්ම පාඩම් වලදී ඉගෙනගත්තා ජාවා තුළ සියලුම දේවල් class ලෙස සලකන බව. ඒ අනුව Errors හා Exception සඳහාද ජාවා තුළ class 2ක් තිබෙනවා.

Error class

මෙම ආකාරයේ errors හඳුන්වන්නේ abnormal errors හෙවත් අස්වාභාවික errors ලෙසින්. වැඩසටහනක් ලිවීමේදී අප එම වැඩසටහන තුළින් සාමාන්‍යයෙන් මෙම ආකාරයේ errors අපේක්ෂා කරන්නේ නැ එනම් ඒවා handle කිරීමට යන්නේ නැ.  
මේ සඳහා උදාහරණ.

- Virtual machine එකේ දෝශ
- ThreadDeath errors

Exception class

ජාවා වැඩසටහන් තුළ Error handle කරනවා යනු මෙම exceptions නියමාකාරයෙන් handle කිරීමයි. මෙහි වැදගත්කම වනුයේ run-time එකේදී කලින් අපේක්ෂා කළ errors ඇතිවුවහොත් program එක terminate වීම හෝ වැරදි ප්‍රතිඵලයක් ලබාදීමෙන් වලක්වා ගැනීමයි.

බහුල වශයෙන් භාවිතයා Exception කිහිපයකට උදාහරණ

- ArithmeticException
- ArrayIndexOutOfBoundsException
- IOException
- FileNotFoundException
- NumberFormatException
- StringOutOfBoundsException

දැන් අපි බලමු ක්‍රමලේඛයක් තුළදී Exception handle කරන්නේ කියලා.  
මේ සඳහා වීඩි 2ක් තිබෙනවා.

### 1) Throw Out (Ignore)

මෙහිදී යම් අපේක්ෂා කළ Exception එකක් භටගතහොත් එය ඉවත දැමීම නොහොත් නොසලකා හැරීම සිදුකෙරෙනවා. ඇතැම් අවස්ථා වලදී මෙම ක්‍රමය යොදාගන්නත් මෙය සුදුසුම ක්‍රමය නොවේ.

දැන් අපි බලමු මෙම ක්‍රමය භාවිතා කරන අයුරු.

Syntax:

```
methodName throws ExceptionName{
 ..method body
}
```

උදා:

1. import java.io.\*;
- 2.
3. class ExceptionEx1{
4. public static void main(String args[]) throws IOException{
5. InputStreamReader isr=new InputStreamReader(System.in);
6. BufferedReader br=new BufferedReader(isr);
7. System.out.print("Enter your name : ");
8. String name=br.readLine();
9. }
10. }

### 2) Catch and handle

සලකනු ලබන code segment එකක් තුළ යම් අපේක්ෂිත Exception එකක් පැනනැගුනහොත් සිදුකළයුතු ක්‍රියාමාර්ගය පිළිබඳ මෙහිදී සඳහන් කෙරේ ජාවා වැඩසටහනක Exception handle කිරීම සඳහා මෙම ක්‍රමය ඉතාම සුදුසු වේ.

Syntax:

```
try{

 ...code segment...

}catch(ExceptionName1 exObj){

 ..handling error1..

}catch(ExceptionName2 exObj){
```

```

.....
..handling error2..
.....
} catch (ExceptionName exObj) {
 ..handling errorn.

} finally {

 ..final action....

}

```

උදා:

```

1. import java.io.*;
2.
3. class ExceptionEx2{
4. public static void main(String args[]) {
5. int age=0;
6. InputStreamReader isr=new InputStreamReader(System.in);
7. BufferedReader br=new BufferedReader(isr);
8. System.out.print("Enter your age : ");
9. try{
10. String ageStr=br.readLine();
11. age=Integer.parseInt(ageStr);
12. } catch (IOException ex1){
13. System.out.println("I/O Error! "+ ex1.getMessage());
14. } catch (NumberFormatException ex2){
15. System.out.println("Invalid number format! "+ ex2.getMessage());
16. }
17. }
18. }

```

output:

```

Enter your age : 2x
Invalid number format! For input string: "2x"

```

Errors catch කිරීමේදී ඒවා අපේක්ෂිත අනුපිළිවෙලට සිදුකළයුතු බව මතකතබා ගත යුතුය. මෙහිදී exceptions ඇතිවිය හැකි අනුපිළිවෙල මෙන්ම class hierarchy එකද වැදගත් වේ. class hierarchy එකක් පහලින් ඇති errors මුලින් catch කර ඉහල ඇති ඒවා පසුව catch කිරීම සිදුකළයුතුය. class hierarchy එක බලාගැනීමට java documentation එක පරිශීලනය කළයුතුවේ.

**finally** block එක පිළිබඳව

මෙතුල අන්තර්ගත කරන code segments අනිවාර්යයෙන්ම execute කිරීම සිදුවේ(Exception එකක් හටගත්තත් නැත්නම්). finally statement එක යෙදෙන අවස්ථා සඳහා උදාහරණ ලෙස open කරන ලද stream එකක් close කිරීම වැනි අවස්ථා දැක්විය හැකිය.

Exception class එකෙහි ඇති getMessage () method එක පිළිබඳව

මෙමගින් Exception එකට අදාල වැඩිදුර තොරතුරු String එකක් ලෙස return කරයි.

අපි දැන් බලමු අපිවිසින් අපේම(our own) Exception සාදන ආකාරය සහ ඒවා throw කරන්නේ කවර ආකාරයෙන්ද කියලා.

මෙහිදී මුලිකවම සිදුකරන්නේ Exception නම් predefined class එක extend කරමින් නව class එකක් සෑදීමයි.

උදා: OwnExceptionDemo.java

```

1. /**
2. *class : OwnExceptionDemo
3. *Author : Kanishka Dilshan
4. *Purpose: Demonstrate howto write custom exceptions
5. *Blog : http://javaxclass.blogspot.com
6. */
7.
8. class Printer{
9. /*assumption:
10. *printing 1 page will decrease the ink level by 1
11. */
12. int inkLevel;//minimum=0
13. int pages;
14.
15. public Printer(int p,int inkLevel){
16. pages=p;
17. this.inkLevel=inkLevel;
18. }
19.
20. public void print(){
21. try{
22. if(inkLevel==0){
23. throw new EmptyCartridgeException();
24. }else if(pages==0){
25. throw new NoPagesException();
26. }else{
27. pages-=1;
28. inkLevel-=1;
29. System.out.println("Page completed.");

```

```

30. }
31. }catch(EmptyCartridgeException ecEx){
32. System.err.println(ecEx.getMessage());
33. }catch(NoPagesException npEx){
34. System.err.println(npEx.getMessage());
35. }
36. }
37. }
38.
39. class NoPagesException extends Exception{
40. NoPagesException(){
41. super("No pages in the paper tray!");
42. }
43. }
44.
45. class EmptyCartridgeException extends Exception{
46. EmptyCartridgeException(){
47. super("Cartridge is empty. Please refill it");
48. }
49. }
50.
51. class OwnExceptionDemo{
52. public static void main(String args[]){
53. //creating Printer object and calling print() method
54. System.out.println("-----p1 Printer object-----");
55. Printer p1=new Printer(5,100);
56. for(int i=0;i<8;i++) //this will cause NoPagesException
57. p1.print();
58.
59. System.out.println("\n-----p2 Printer object-----");
60. Printer p2=new Printer(5,4);
61. for(int i=0;i<5;i++) //this will cause EmptyCartridgeException
62. p2.print();
63. }
64. }

```

මෙම වැඩසටහනෙහි output එක පහත පරිදි වේ.

```

-----p1 Printer object-----
Page completed.
Page completed.
Page completed.
Page completed.
Page completed.
No pages in the paper tray!
No pages in the paper tray!
No pages in the paper tray!

-----p2 Printer object-----
Page completed.
Page completed.
Page completed.
Page completed.
Cartridge is empty. Please refill it

```

දැන් ඔබට Error handling පිළිබඳව යම් වැටහීමක් ඇතැයි සිතමි. වැඩිදුරටත් අභ්‍යාසයේ යෙදීමෙන් තවදුරටත් එය තහවුරු කරගැනීමට උත්සාහ ගන්න. ගැටලු සහගත තැන් ඇත්නම් ඒවා ඉදිරිපත් කරන්න.

මිලහ පාඩමෙන් අපි Threads පිළිබඳ අධ්‍යයනය කරමු.

# ජාවා යොදාගෙන බහුකාඨී වැඩසටහන් ලිවීම (Java Threads) I

Friday, September 10, 2010 by Kaniskha Dilshan

පරිගණක වැඩසටහනක් සැලසුම කිරීමේදී එය මගින් යම් යම් ක්‍රියාවන්(methods) එකවිට(parallel) සිදුකළයුතු අවස්ථා අපට මුණගැසේ. Multitasking සඳහා උදාහරණයක් ලෙස web browser එකක tabs කිහිපයක් එකවිට load වීම ගත හැකිය. එවැනි අවස්ථාවන් ක්‍රියාවට නැංවීම (implement) සඳහා multitasking සංකල්පය භාවිතයට ගත හැකිය.

Multitasking වැඩසටහන්හි පහත අයුරින් ප්‍රධාන ආකාර 2ක් දැකිය හැකිය.

## \* Process Based

මෙහිදී සිදුවනුයේ පරිගණකයේ වැඩසටහන් කීපයක් සමගාමීව ක්‍රියාකිරීමෙන් අදාළ ප්‍රතිඵලය ලබාදීමයි. process අතර සන්නිවේදනය(දත්ත හුවමාරුව) පහසු නොවේ. ඒ සඳහා pipes, files, sockets, shared memory ආදී විවිධ උපක්‍රම යොදාගැනීමට සිදුවේ.

## \* Thread Based

මෙහිදී සිදුවනුයේ එක් වැඩසටහනක් මගින් කාඨිකත් කිහිපයක් සමගාමීව සිදුකර අදාළ ප්‍රතිඵලය ලබාදීමයි. මෙහිදී අදාළ වැඩසටහන තුළ Threads ලෙසින් පවතින කාඨිකත් කිහිපය කලමණාකරණය කිරීම මගින් ජරතීඵලය ලබාගැනේ. සියලුම threads මගින් එකම address space එක share කරගන්නා නිසා threads අතර සන්නිවේදනය සාපේක්ෂව පහසුය.

අපි මෙහිදී සලකනුයේ Java යොදාගෙන multitasking සිදුකිරීමයි. මුලින් සඳහන් කළ multitasking අකාරය හෙවත් process based multitasking ජාවාහි පාලනයෙන් ඔබ්බෙන් පවතින නිසා ජාවාතුළ multitasking සිදුකරනුයේ threads යොදාගැනීමෙනි. Process based multitasking සිදුකිරීම C/C++ වැනි භාෂා යොදාගැනීමෙන් සිදුකළ හැක.

ඕනෑම ජාවා වැඩසටහනක මුලින්ම ආරම්භ වන thread එක වනුයේ main thread එකයි.

ජාවා තුළ මූලිකව threads ප්‍රභේද 2ක් පවතී. එනම් **Thread** class එක සහ **Runnable** interface එකයි. එම නිසා ජාවාහි Threads සෑදිය හැකි මූලිකම ක්‍රම දෙක වනුයේ

1. **java.lang.Thread** class එක Extend කර **run()** method එක override කිරීම
2. **java.lang.Runnable** interface එක implement කර **run()** method එක implement කිරීම.

Note : extend කිරීම සහ interfaces පිළිබඳ පෙර පාඩම් නැවත මතක් කරගැනීමට අවශ්‍ය නම් පහත සබැඳි භාවිතා කරන්න.

[ජාවා interface සංකල්පය සහ එහි යෙදීම්](#)

[ජාවා වස්තු පාදක ක්‍රමලේඛනය VI \(Inheritance\) :: 1 කොටස](#)

**Thread** class එක extend කිරීම

TestThread.java

```
class TestThread{
 public static void main(String args[]){
 MyTask mtl=new MyTask();
 mtl.start(); //start the thread
 }
}
```

class MyTask extends Thread {

```
 public void run(){
 //do some task
 }
}
```

**Runnable** interface එක implement කිරීම

TestThreadRunnable.java

```
class TestThreadRunnable{
 public static void main(String args[]){
 Thread t1=new Thread(new MyTask());
 t1.start(); //start the thread
 }
}
```

class MyTask implements Runnable {

```
 public void run(){
 //do some task
 }
}
```

අපි ඊළඟ පාඩමෙන් threads පිළිබඳ තවදුරටත් හදාරමු.

# ජාවා යොදාගෙන බහුකාඩී වැඩසටහන් ලිවීම (Java Threads) II

Saturday, September 11, 2010 by Kaniskha Dilshan



අපි මීට පෙර පාඩමේදී සාකච්ඡා කලා Threads හා සම්බන්ධ මූලිකම මූලධර්ම පිළිබඳව. අපි මෙම පාඩමෙන් බලමු Threads යොදාගෙන වැඩසටහන් ලියන ආකාරය. Threads යොදාගෙන ජාවා වැඩසටහන් ලිවීමේදී ඉතා වැදගත් වන methods කිහිපයක් තිබෙනවා අපි මුලින්ම එම methods පිළිබඳ සලකා බලමු. ඕනෑම Thread එකක් සැලකීමේදී එය පවත්නා තත්වයන් කිහිපයක් ඇත. පහත Thread state diagram එක මගින් එම අවස්ථා පිළිබඳ සහ thread එකක හැසිරීම පිළිබඳ අධ්‍යයනය කරමු.

© javaxclass.blogspot.com

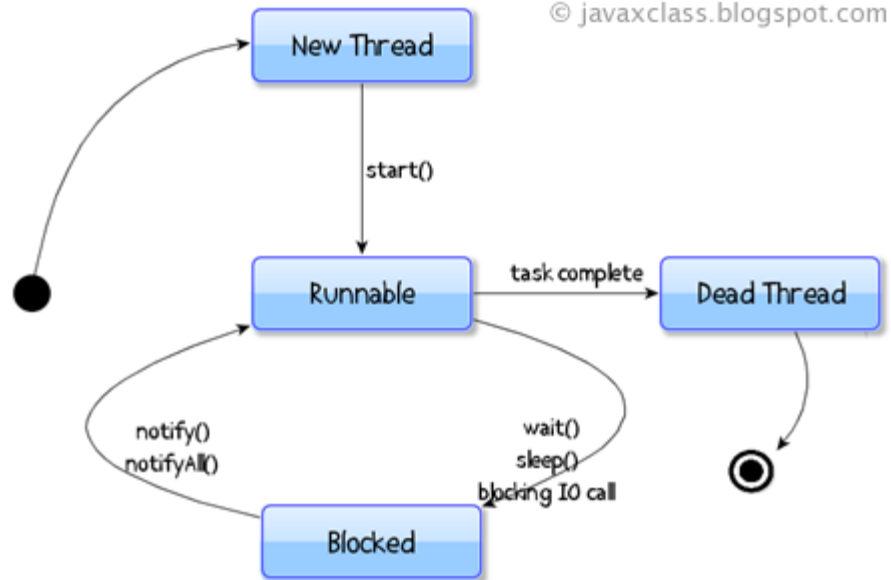


Fig 1.0 Thread State Diagram

| Method Name            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Thread.sleep()         | දෙන ලද කාලයක් සඳහා අවශ්‍ය thread එකක් නවතා දැමිය හැක (pause).<br>syntax:<br>sleep(long milliseconds)<br>sleep(long ms,long nanosec)<br>eg:<br>try{<br>Thread.sleep(1000);<br>}catch (InterruptedException e){<br>System.err.println("Operation was Interrupted!");<br>}                                                                                                                                                                                                                                                              |
| Thread.yield()         | අනෙකුත් threads සඳහා execute වීමට ඉඩ ලබාදේ. Thread synchronization පිළිබඳ සාකච්ඡා කිරීමේදී මේ ගැන වැඩිදුරටත් අධ්‍යයනය කරමු.                                                                                                                                                                                                                                                                                                                                                                                                          |
| Thread.start()         | අදාල Thread එක ආරම්භ කෙරේ. එනම් thread එක runnable state එකට පත්කරයි.                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| Thread.isAlive()       | දෙන ලද thread එකක් ජීවී තත්වයේ පවතීදැයි පරීක්ශා කෙරේ. Thread එක ජීවී තත්වයේ පවති නම් true ලෙසටද නැතහොත් false ලෙසටද return කෙරේ. Thread එකක් live තත්වයේ පැවතීමට නම් එය start වී තිබීම හා එය dead state එකේ නොතිබීම අනිවාර්ය වේ.                                                                                                                                                                                                                                                                                                     |
| Thread.activeCount()   | Active threads ප්‍රමාණය return කරයි.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Thread.enumerate()     | මේ වන විට active තත්වයේ පවතින සියලු threads දෙන ලද array එකකට fill කිරීම සිදුකරයි.                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Thread.currentThread() | Current thread එක return කරයි.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| Thread.dumpStack()     | ජාවා වැඩසටහන් debug කිරීමේදී වැදගත් වේ.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| Thread.setPriority()   | කිසියම් Thread එකකට ලබාදෙන priority එක int value එකක් ලෙස ලබාදේ. (1 සිට 10 දක්වා)                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Thread.getId()         | දෙනලද thread එකක් හඳුනාගැනීම සඳහා ලැබී ඇති ID එක long value එකක් ලෙස return කරයි.                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Thread.setDaemon()     | කිසියම් thread එකක් daemon ස්වභාවයට පත් කරයි. Daemon thread එකක් යනු user defined threads සඳහා සහය දැක්වීම සඳහා පසුබිමින් ධාවනය වන(run) threads වේ. JVM එකේ Garbage collector එක මීට උදාහරණයකි.<br><br>මෙම method එක මගින් අප විසින් ලියන ලද කිසියම් thread එකක් daemon nature එකට පත්කල හැක. සාමාන්‍යයෙන් daemon thread එකකට ලැබෙනුයේ low priority එකකි. නමුත් අපට අවශ්‍ය නම් daemon threads සඳහා වැඩි priority එකක් ලබාදිය හැක.<br><br>මෙහිදී විශේෂයෙන් සැලකිය යුතු කාරණය වන්නේ ඕනෑම daemon thred එකක් ආරම්භ වීමෙන් පසු එහි daemon |



|                  |                                                                                                                                                                                                                                                                                          |
|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                  | ස්වභාවය වෙනස් කළ නොහැකිය යන්නයි. නමුත් එහි priority එක වෙනස් කළ හැක.<br><br>eg:<br>MyThread mt=new MyThread();<br>mt.setDaemon(true);<br>mt.start();                                                                                                                                     |
| Thread.run()     | Thread එකක වැදගත්ම method එකක් ලෙස සැලකිය හැක. ඕනෑම thread එකක මෙම method එක override කර තිබිය යුතුය. start() method එක call කිරීමෙන් පසු execute කිරීමට පටන්ගන්නේ මෙම run() method එක තුළ ඇති විධානයන් වේ.                                                                              |
| Thread.destroy() | කිසිදු cleanup එකකින් තොරව දෙනලද thread එකක් destroy කරදමයි. මෙම method එක deadlocks ඇතිවීමට හේතුවන නිසා මෙය deprecated method එකක් ලෙස දක්වා ඇත. Deadlocks පිළිබඳ ඉදිරි පාඩමකදී සාකච්ඡා කෙරේ.                                                                                           |
| Thread.stop()    | කිසියම් thread එකක් stop කිරීමට යොදාගැනේ. මෙයද unsafe operation එකක් ලෙස සැලකේ. එම නිසා මෙය deprecated method එකක් ලෙස දක්වා ඇත. Thread එකක් නැවැත්වීම සඳහා stop method එක භාවිතා නොකර වෙනත් technique එකක් භාවිතා කර ආරක්ෂිත ආකාරයකට threads නවතා දමන අයුරු අපි ඉදිරියේදී සාකච්ඡා කරමු. |

ජාවා භාවිතයෙන් thread create කරන අයුරු ආදර්ශනය කිරීමට ලියන ලද පහත සරල වැඩසටහන හොඳින් අධ්‍යයනය කරන්න.

ThreadLauncher.java

```

1. /**
2. *class : ThreadLauncher
3. *Author : Kanishka Dilshan
4. *Purpose: Showing how to create threads in Java
5. *Blog : http://javaxclass.blogspot.com
6. */
7.
8. class ThreadLauncher{
9. public static void main(String args[]){
10. MyThread mt1=new MyThread("motor1");
11. MyThread mt2=new MyThread("motor2");
12. mt1.start();
13. mt2.start();
14. }
15. }
16.
17. class MyThread extends Thread{
18. String name;
19. public MyThread(String name){
20. this.name=name;
21. }
22.
23. public void run(){
24. System.out.println("Thread : "+name+" started!");
25. for(int i=0;i<5;i++){
26. System.out.println("I am "+name+" Thread.");
27. try{
28. this.sleep(1000);
29. }catch(InterruptedException e){
30. System.err.println("Thread was interrupted! "+ e.getMessage());
31. }
32. }
33. System.out.println("Thread "+name+" is going to die!");
34. }
35. }
```

Code 1.0

Output:

```

Thread : motor1 started!
Thread : motor2 started!
I am motor1 Thread.
I am motor2 Thread.
I am motor2 Thread.
I am motor1 Thread.
I am motor1 Thread.
I am motor2 Thread.
I am motor1 Thread.
I am motor2 Thread.
I am motor1 Thread.
I am motor2 Thread.
Thread motor2 is going to die!
Thread motor1 is going to die!
```

Fig 2.0 Output of code 1.0

Threaded program එකක out put එක ඉහත පරිදි වේ. mt1 හා mt2 යනු threads 2යි. අපි එම වැඩසටහනම mt1 හා mt2 යනු runnable නොවන objects ලෙස සකස් කළහොත් output එක පහත පරිදි වේ.

```

Thread : motor1 started!
I am motor1 Thread.
I am motor1 Thread.
I am motor1 Thread.
I am motor1 Thread.
I am motor1 Thread.
Thread motor1 is going to die!
Thread : motor2 started!
I am motor2 Thread.
I am motor2 Thread.
I am motor2 Thread.
I am motor2 Thread.
I am motor2 Thread.
Thread motor2 is going to die!
```

ඔබට threads පිළිබඳ යම් අවබෝධයක් දැන් ලැබී තිබිය යුතුය. අපි ඉදි පාඩම් වලින් තවදුරටත් threads පිළිබඳ සාකච්ඡා කරමු.

# Java සමග Software Design Patterns (1 කොටස)

Wednesday, October 6, 2010 by Kaniskha Dilshan



කිසියම් පරිගණක භාෂාවක් යොදාගෙන මෘදුකාංග නිපදවීමේදී අප විසින් කිසියම් සැලසුමක් මුලින් සකස් කළ යුතු වනවා. මෙහිදී අදාළ සැලැස්ම ඉතා හොඳ එකක් වීම ඉතා වැදගත්. හොඳ මෘදුකාංග සැලසුමක් නිසා කේතකරණය මෙන්ම මෘදුකාංගයේ නව යාවත් කාලීන කිරීම් පුලුල් කිරීම ආදිය පහසු වීම සිදුවනවා.

හොඳ design එකක් සාදාගැනීම සඳහා භාවිතාකරන එක තාක්ෂණයක් නමයි software design patterns කියලා කියන්නේ. කිසියම් මෘදුකාංගයක් සැලකීමේදී එහි design pattern 1කට වැඩි සංඛ්‍යාවක් තිබීමට පුලුවන්.

software design pattern සඳහා වන [Wikipedia](#) හි අර්ථ දැක්වීම

*"A general reusable solution to a commonly occurring problem in software design"*

සරළව විස්තර කරන්නේ නම්. design pattern එකක් යනු විවිධ අවස්ථාවන් සඳහා යොදාගත හැකිවන පරිදි කිසියම් ගැටලුවක් විසඳන ආකාරයයි. එම නිසා ඕනෑම software pattern එකක් සැලසුම වීම ඒය design pattern එකක් නොවිය හැකියි.

උදා: algorithm(sorting,encrypting,compressing..etc) සහ data structures(queues, linked lists, stacks...etc) යන ඒවා design patterns ලෙස නොසැලකේ.

පහත දැක්වෙන ආකාරයට design patterns වර්ග කල හැකිය

- creational design

object සෑදීම විධිමත් ලෙස කලමනාකරණය කිරීමට යොදාගැනේ

- structural design

විධිමත් ලෙස objects එකලස් කිරීම සඳහා යොදාගැනේ

- behavioral design patterns

flow control සහ algorithms කලමණාකරනය කිරීම සඳහා යොදාගැනේ

අපි දැන් බලමු singleton design pattern එක පිළිබඳව.

Singleton Design Pattern



මෙය **creational pattern** ආකාරයට අයත් වේ. සාමාන්‍යයෙන් class එකක් සැලසුම වට object 1ක් හෝ ඊට වැඩි ගනණක් සෑදිය හැකිය. class එකකින් instance එකක් පමණක් ලබාගන්නා අන්දම මෙමගින් පැහැදිලි කෙරේ.

class structure

| Singleton                         |
|-----------------------------------|
| -static uniqueInstance: Singleton |
| +static getInstance(): Singleton  |
| #<<constructor>> Singleton()      |

මෙහිදී Singleton class එක සඳහා subclasses නොමැති නම් Singleton constructor එක private ලෙස දැක්විය යුතුය.

මෙහිදී Singleton එක access කළ හැක්කේ getInstance() static method එක හරහා වේ. එනම් Singleton එකෙන් object create කිරීම කළ හැක්කේ getInstance() method එක හරහා පමණි.

Sample Code:

```
1. /**
2. *class : Singleton
3. *Author : Kanishka Dilshan
4. *Purpose: Showing how to implement Singleton design pattern
5. *Blog : http://javaxclass.blogspot.com
6. */
7.
8. class Singleton {
9. private static Singleton instance=null;
10.
11. protected Singleton(){
12.
13. }
14.
15. public static Singleton getInstance(){
16. if(instance==null){
17. instance=new Singleton();
18. }
19. return instance;
20. }
21. }
```

දැන් අපි බලමු Singleton design pattern එක යොදාගෙන සරල වැඩසටහනක් ලියන අයුරු.

```
1. /**
2. *class : PrinterSingleton
3. *Author : Kanishka Dilshan
4. *Purpose: Showing how to use Singleton design pattern
5. *Blog : http://javaxclass.blogspot.com
6. */
7.
8. class PrinterSingleton {
9. private static PrinterSingleton instance=null;
10. int jobsCount;
11. private PrinterSingleton(){
12. jobsCount=0;
13. }
14.
15. public static PrinterSingleton getInstance(){
16. if(instance==null){
17. instance=new PrinterSingleton();
18. }
19. return instance;
20. }
21.
22. public void addNewJob(){
23. jobsCount++;
24. }
25.
26. public int getJobsCount(){
27. return jobsCount;
28. }
29. }
30.
31.
32. public class SingletonTest {
33. public static void main(String args[]){
34. PrinterSingleton p1=PrinterSingleton.getInstance();
35. PrinterSingleton p2=PrinterSingleton.getInstance();
36.
37. p1.addNewJob();
38. p1.addNewJob();
39. p1.addNewJob();
40.
41. System.out.println("p1 job count "+p1.getJobsCount());
```

```

42. System.out.println("p2 job count "+p2.getJobsCount());
43.
44. if(p1.equals(p2)){
45. System.out.println("p1 and p2 are same object");
46. }
47. }
48. }

```

output:

```

p1 job count 3
p2 job count 3
p1 and p2 are same object

```

Singleton design pattern එක වැඩිදියුණු කරමින් Coupletion design එකක්(exactly 2 instances) implement කරන අයුරු මිලඟට බලමු.

```

1. /**
2. *class : CoupletionTest, Coupletion
3. *Author : Kanishka Dilshan
4. *Purpose: Showing how to implement coupleton design pattern
5. *Blog : http://javaxclass.blogspot.com
6. */
7.
8. class Coupletion{
9. private static Coupletion _instance1=null;
10. private static Coupletion _instance2=null;
11. private String ins_name;
12. private Coupletion(String insName){
13. this.ins_name=insName;
14. }
15.
16. public static Coupletion getInstance(int instanceNo,String ins_name){
17. if(instanceNo==1){
18. if(_instance1==null){
19. _instance1=new Coupletion(ins_name);
20. }
21. return _instance1;
22. }else if(instanceNo==2){
23. if(_instance2==null){
24. _instance2=new Coupletion(ins_name);
25. }
26. return _instance2;
27. }else{
28. return null;
29. }
30. }
31.
32. public String getInsName(){
33. return ins_name;
34. }
35.
36. }
37.
38. public class CoupletionTest{
39. public static void main(String args[]){
40. Coupletion c1=Coupletion.getInstance(1,"My Coupletion1");
41. Coupletion c2=Coupletion.getInstance(2,"My Coupletion2");
42. Coupletion c3=Coupletion.getInstance(1,"My CoupletionX");
43. Coupletion c4=Coupletion.getInstance(2,"My CoupletionY");
44. System.out.println("c1 instance name : "+c1.getInsName());
45. System.out.println("c2 instance name : "+c2.getInsName());
46. System.out.println("c3 instance name : "+c3.getInsName());
47. System.out.println("c4 instance name : "+c4.getInsName());
48. }
49. }

```

output:

```

c1 instance name : My Coupletion1
c2 instance name : My Coupletion2
c3 instance name : My Coupletion1
c4 instance name : My Coupletion2

```

අපි ඉදිරි පාඩම වලදී තවත් design patterns පිළිබඳ සාකච්ඡා කරමු.

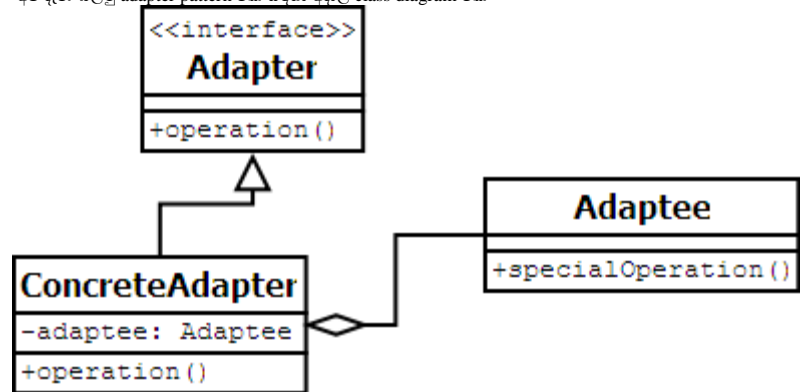
# Java සමග Software Design Patterns (2 කොටස | Adapter Pattern)

Wednesday, October 6, 2010 by Kanishka Dilshan



Adapter නම් design pattern එක structural ආකාරයේ pattern එකකි. ලියන ලද කිසියම් program code එකක් දැනට පවතින වෙනත් නොගැලපෙන(incompatible) code-base එකකට(උදා: google maps API, twitter API) සම්බන්ද කිරීමේදී මෙම design pattern එක apply කළ හැකිය. මෙහිදී සිදුකරනුයේ විධිමත් ක්‍රමයකට අපේ program එකට පරිහානි ර code-base එක call කිරීමට class structure එකක් හැදීමයි. සාමාන්‍යයෙන් අප විසින් සකසන මෙම class තුලදී process එකක් නොපවති ඒ වෙනුවට එම class මගින් සිදුකරනුයේ අදාල කාණ්ඩය සිදුකරගැනීම පිණිස පවතින code-base එක call කිරීමයි.

අපි දැන් බලමු adapter pattern එක සඳහා අදාළ class diagram එක



ඔබට UML පිළිබඳ යම් අවබෝධයක් ඇත්නම් ඉහත class diagram එක මගින් adapter pattern එකෙහි ව්‍යුහය පිළිබඳ මනා වැටහීමක් ලැබෙනු ඇත. ඔබට UML(Unified Modeling Language) පිළිබඳ අවබෝධයක් නොමැතිනම් [මෙම tutorial එක](#) භාගත කරගෙන හොඳින් අධ්‍යයනය කරන්න. ඉන්පසු ඔබට ඉතා පහසුවෙන් මෙම UML diagram එක සහ ඉදිරි පාඩම් වලදී ඉදිරිපත් කෙරෙන UML diagrams වටහාගත හැකිවනු ඇත.

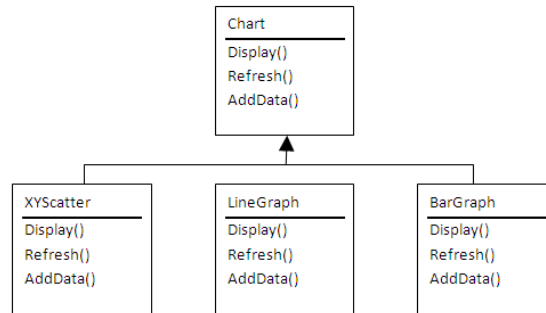
මෙය කවදුරටත් වටහාගැනීම පිණිස මෙම pattern එක ජාවා තුල implement කරන ආකාරය සලකා බලමු.

```
1. /**
2. *class : AdapterTest
3. *Author : Kanishka Dilshan
4. *Purpose: Showing how to implement Adapter design pattern
5. *Blog : http://javaclass.blogspot.com
6. */
7.
8.
9. interface Adapter {
10. void operation();
11. }
12.
13. class ConcreteAdapter implements Adapter {
14. private Adaptee adaptee ;
15.
16. public ConcreteAdapter(){
17. adaptee=new Adaptee();
18. }
19.
20. public void operation(){
21. adaptee.specialOperation();
22. }
23. }
24.
25. class Adaptee {
26. //this is the external codebase
27. void specialOperation(){
28. System.out.println("Special operation was launched!");
29. }
30. }
31.
32. //test class
33. public class AdapterTest{
34. public static void main(String args[]){
35. ConcreteAdapter ca=new ConcreteAdapter();
36. ca.operation();
37. }
38. }
```

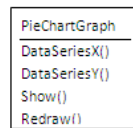
මෙම pattern එක වඩාත් හොඳින් තහවුරු කරගැනීම සඳහා ගැටලුවක් සාකච්ඡා කරමු. මෙම ගැටලුව උපුටාගන්නේ ශ්‍රී ලංකා තොරතුරු තාක්ෂණ ආයතනයේ software engineering II සඳහා ලබාදුන් tutorial එකකිනි.

ගැටලුව:

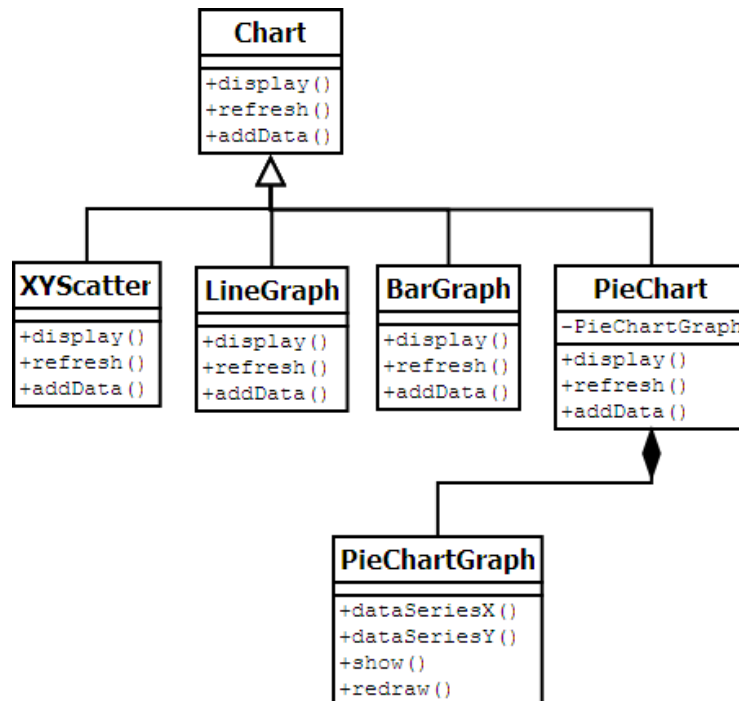
You have developed a set of classes that draws graphs.



Here the base class is the Chart Class. You need to develop a PieChart class. There is an existing PieChart class called PieChartGraph (See below). You want to connect your code base to this new Class. Briefly describe how an appropriate design pattern can be used for this purpose.



පිළිතුර:



```

1. abstract class Chart {
2. abstract protected void display();
3. abstract protected void refresh();
4. abstract protected void addData();
5. }
6.
7. class XYScatter extends Chart {
8. public void display(){
9. System.out.println("Displaying XYScatter..");
10. }
11.
12. public void refresh(){
13. System.out.println("Refreshing XYScatter..");
14. }
15.
16. public void addData(){
17. System.out.println("Adding data to the XYScatter..");
18. }
19. }
20.

```

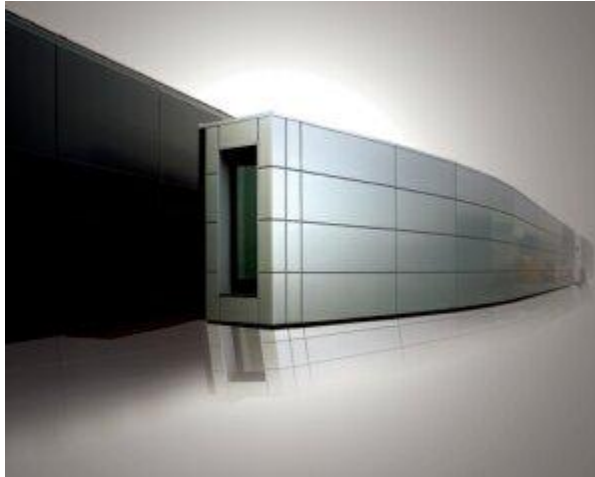
```

21. class LineGraph extends Chart {
22. public void display(){
23. System.out.println("Displaying LineGraph..");
24. }
25.
26. public void refresh(){
27. System.out.println("Refreshing LineGraph..");
28. }
29.
30. public void addData(){
31. System.out.println("Adding data to the LineGraph..");
32. }
33. }
34.
35. class BarGraph extends Chart {
36. public void display(){
37. System.out.println("Displaying BarGraph..");
38. }
39.
40. public void refresh(){
41. System.out.println("Refreshing BarGraph..");
42. }
43.
44. public void addData(){
45. System.out.println("Adding data to the BarGraph..");
46. }
47. }
48.
49. class PieChart extends Chart {
50.
51. private PieChartGraph pcg;
52.
53. public PieChart(){
54. pcg=new PieChartGraph();
55. }
56.
57. public void display(){
58. System.out.println("Displaying PieChart..");
59. pcg.show();
60. }
61.
62. public void refresh(){
63. System.out.println("Refreshing PieChart..");
64. }
65.
66. public void addData(){
67. System.out.println("Adding data to the PieChart..");
68. }
69. }
70.
71. class PieChartGraph{
72.
73. public void dataseriesX(){
74.
75. }
76.
77. public void dataseriesY(){
78.
79. }
80.
81. public void show(){
82. System.out.println("Showing PieChartGraph..");
83. }
84.
85. public void redraw(){
86. System.out.println("Refreshing PieChartGraph..");
87. }
88. }
89.
90. public class Answer {
91. public static void main(String args[]){
92. PieChart pc=new PieChart();
93. pc.display();
94. }
95. }

```

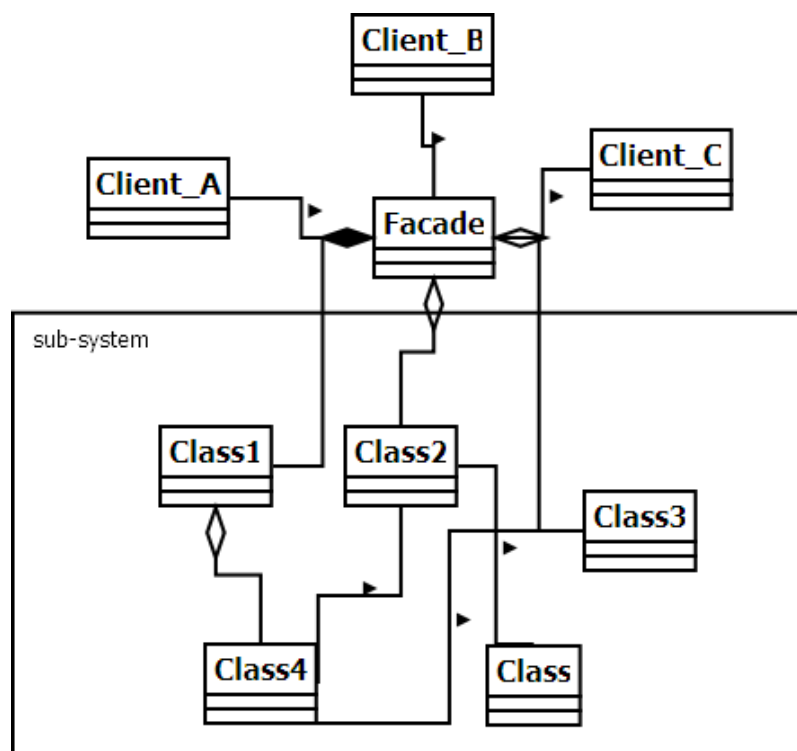
# Java සමග Software Design Patterns (3 කොටස | Facade Pattern)

Thursday, October 7, 2010 by Kaniskha Dilshan



ෆැසාඩ් | Facade යනු structural ආකාරයේ design pattern එකකි . මෙමගින් කිසියම් සංකීර්ණ උප පද්ධතියක(sub system) lower level classes invoke කිරීම මගින් යම් කාර්යයක් සිදුකිරගැනීම සඳහා higher level interface එකක් සපයනු ලබයි. මෙලෙස facade pattern එක යොදවීමත් කිසියම් පද්ධතියක් සැලසුම් කිරීම නිසා එම පද්ධතියේ **coupling** යන ගුණය අඩුවීම සිදුවේ එනම් එම පද්ධතියේ ස්ථායීතාවය කෙරෙහි එය සුභදායක ලෙස බලපායි මීට අමතරව .coupling ගුණය අඩුවීම යම් පද්ධතියක කාර්යක්ෂමතාවය වැඩිවීම පිනිසද හේතුවේ යම් .system එකකට facade pattern එක apply කල පසු එතුල පවතින සංකීර්ණ sub system එකක් කිසියම් client class කිහිපයක් මගින් access කරනුයේ facade එක හරහාවේ . එමගින් ඉතා විශාල වාසි එම පද්ධතියට අත්වේ ඉන් එක් වාසියක් නම් client ගේ වෙනසක් නොවන පරිදි සංකීර්ණ sub system එක තුල වෙනස්කම් සිදුලඟැකි විමයිඑමගින් අප වි .යින් සැලසුම් කරන පද්ධතිය අඩුල් ජාලයක් වීම වලකනු ලබයි.

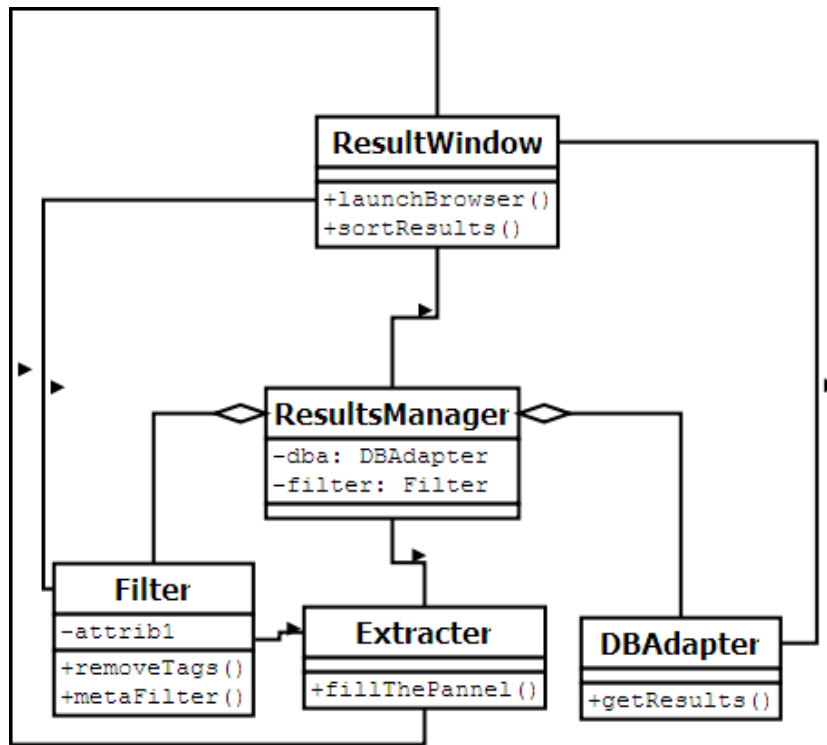
Facade හි ව්‍යුහය



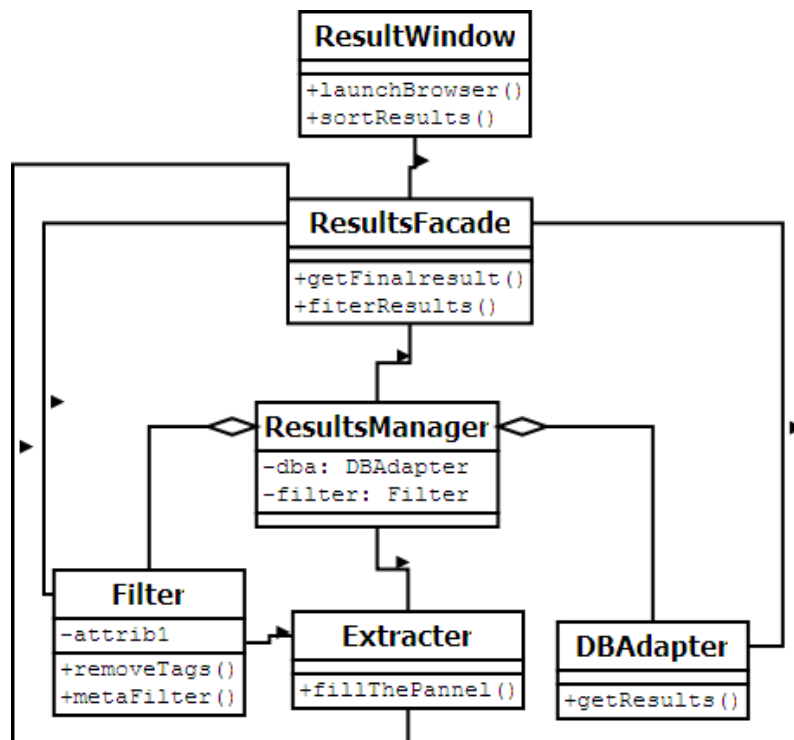
මෙම pattern එක තවදුරටත් වටහා ගැනීම පහසුවීම සඳහා අපි දැන් Facade එක apply කිරීමට පෙර class diagram එකක් සහ Facade design pattern එක apply කිරීමෙන් පසුව ලැබෙන විසඳුමකට අදාල class diagrams සලකා බලමු.

Facade යෙදීමට පෙර





Facade යෙදූ පසු



2වන class diagram එක අධ්‍යයනය කිරීමේදී coupling ගුණාංග විශාල ලෙස අඩුවී ඇති බව ඔබට වැටහෙනවා ඇති. අපි මීට පෙර සාකච්ඡා කළ [Adapter design pattern](#) එක සහ Facade එක මගින් කෙරෙනුයේ classes wrap කිරීමකි .Facade pattern එකේදී වඩා සංකීර්ණ interface එකක් ලැබේ අපි ඉදිරි පාඩම් වලදී තවත් .design pattern පිළිබඳ අධ්‍යයනය කරමු.

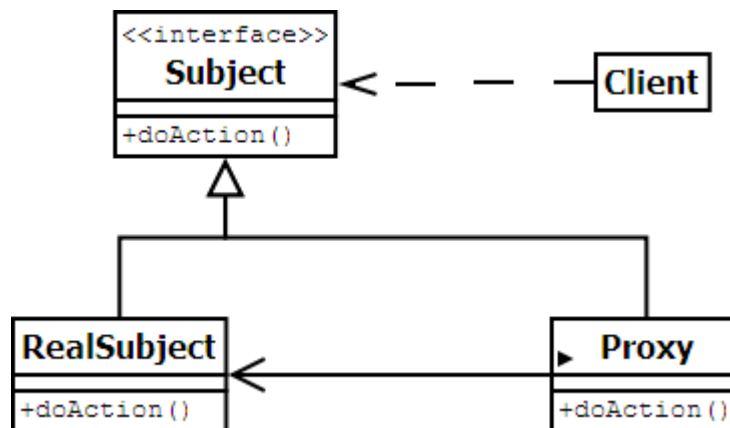
# Java සමග Software Design Patterns (4 කොටස | Proxy Pattern)

Friday, October 8, 2010 by Kanishka Dilshan

බොහෝ පරිගණක වැඩසටහන් සැලසුම් කිරීමේදී අපට පද්ධතියෙන් පරිභාහිර සම්පත්(resources) අප සැලසුම් කරන වැඩසටහන තුළ භාවිතයට ගැනීමට සිදුවේ. උදාහරණ වශයෙන් ජාල සම්බන්ධතා(network connections), files, පරිගණකයේ සසම්භාවී මතකය තුළ පවතින විශාල වස්තුවක්(a large object in memory) හෝ සම්පත් අපතේ යෑම නිසා duplicate කළ නොහැකි අනෙකුත් ඕනෑම resource එකක් සැලකිය හැකිය.

මෙවැනි අවස්ථාවලදී අප සකස්කරන සැලසුම අසාර්ථක එකක් වුවහොත් එය කෙලවර වන්නේ අධික ලෙස bandwidth එක භාවිතා කරන හෝ පරිගණක මතකය අපතේ යවන හෝ කාඩ්ක්ෂමතාව අතින් ඉතාම අඩු මට්ටමක පරිගණක වැඩසටහනකිත්. ප්‍රොක්සි(proxy design pattern) යනු මෙවැනි අවස්ථාවන්හිදී සාර්ථකව apply කළ හැකි design pattern එකකි.

මෙහිදී සිදුකරනුයේ අදාළ resource එක ප්‍රවේශය කිරීම(access) සඳහා interface එකක් සැකසීමයි. පහත ඉදිරිපත් කර ඇති class diagram එක මගින් මෙහි ක්‍රියාකාරීත්වය වටහාගැනීම පහසුය.



අපි දැන් ඉහත model එක ජාවාතුල ක්‍රියාවට නංවන අන්දම බලමු.  
ProxyTest.java

```
1. /**
2. *class : ProxyTest
3. *Author : Kanishka Dilshan
4. *Purpose: Showing how to implement Proxy design pattern
5. *Blog : http://javaxclass.blogspot.com
6. */
7.
8. interface Subject {
9. void doAction();
10. }
11.
12. class RealSubject implements Subject{
13.
14. public RealSubject(){
15. //init resources
16. }
17.
18. public void doAction(){
19. //do something
20. }
21.
22. private void someInternalProcess(){
23. /*do some internal process related to
24. *the external resource
25. */
26. }
27. }
28.
29. class Proxy implements Subject{
30. private Subject subj=null;
31.
32. public Proxy(){
33. //init initial attributes
34. }
35.
36. public void doAction(){
37. //handle the RealSubject
38. if(subj==null){
39. subj=new RealSubject();
```

```

40. }
41. subj.doAction();
42. }
43.
44. }
45.
46. public class ProxyTest {
47. public static void main(String args[]){
48. Subject subj1=new Proxy();
49. Subject subj2=new Proxy();
50.
51. subj1.doAction();
52. //this initializes the resource
53. subj2.doAction();
54. //this also initializes the resource
55. subj1.doAction();
56. //this time it will not initialize the resource
57. }
58. }

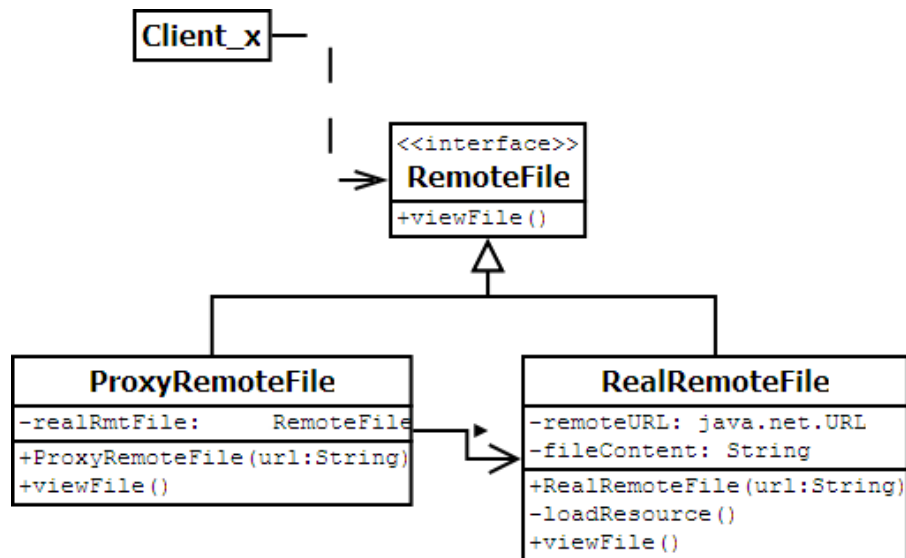
```

ඉහත class diagram එක implement කරන අන්දම වටහා ගැනීමට සහ Proxy හි මූලධර්මය වටහාගැනීමට අදාළ ජාවා කේතය ප්‍රමාණවත් වුවද ප්‍රායෝගික වශයෙන් Proxy pattern එක කිසියම් ගැටලුවකට apply කරන අන්දම වටහාගැනීමට අපි පහත උදාහරණය සලකා බලමු.

ගැටලුව:

URL එක දෙතලද(<http://sites.google.com/site/ansisliit/Home/MyData.dat>) data file එකක් proxy design pattern එක යොදා view කරවීමට අදාළ UML design එක සහ එය ජාවා යොදාගෙන implement කරන්න.

පිළිතුර:



RemoteFile.java

```

1. /**
2. *class : ProxyExampleTest
3. *Author : Kanishka Dilshan
4. *Purpose: Showing how to implement Proxy design pattern
5. *Blog : http://javaclass.blogspot.com
6. */
7. public interface RemoteFile {
8. void viewFile();
9. }

```

RealRemoteFile.java

```

1. import java.io.BufferedReader;
2. import java.io.IOException;
3. import java.io.InputStreamReader;
4. import java.net.MalformedURLException;
5. import java.net.URL;
6. import java.net.URLConnection;
7.
8. public class RealRemoteFile implements RemoteFile {
9. private String fileContent;

```

```

10. private URL url;
11.
12. public RealRemoteFile(String u){
13. try {
14. url=new URL(u);
15. } catch (MalformedURLException e) {
16. System.err.println("Invalid URL!");
17. }
18. fileContent="";
19.
20. loadResource();
21. }
22.
23. @Override
24. public void viewFile() {
25. System.out.println("Displaying the remote file...\n");
26. System.out.println(fileContent);
27. }
28.
29. private void loadResource(){
30. System.out.println("Loading Remote Resource");
31. try {
32. URLConnection conn=url.openConnection();
33. conn.setDoOutput(false); //read only
34. InputStreamReader isr=new InputStreamReader(conn.getInputStream());
35. BufferedReader br=new BufferedReader(isr);
36. String tempLine;
37. while((tempLine=br.readLine())!= null){
38. fileContent+=tempLine+"\n";
39. }
40. } catch (IOException e) {
41. System.err.println("Cannot open the location " + e.getMessage());
42. }
43. }
44. }

```

ProxyRemoteFile.java

```

1. public class ProxyRemoteFile implements RemoteFile{
2. RemoteFile rmtFile=null;
3. String rmtFilePath;
4.
5. public ProxyRemoteFile(String url){
6. rmtFilePath=url;
7. }
8.
9. @Override
10. public void viewFile() {
11. long st1,st2;
12. st1=System.nanoTime();
13. if(rmtFile==null){
14. rmtFile=new RealRemoteFile(rmtFilePath);
15. }
16. rmtFile.viewFile();
17. st2=System.nanoTime();
18. long timeDiff=st2-st1;
19. double timeAmt=timeDiff*10e-12;
20. System.out.println("Time taken to view : "+ timeAmt+ " s");
21. }
22. }

```

ProxyExampleTest.java

```

1. public class ProxyExampleTest {
2. public static void main(String args[]){
3. String location="http://sites.google.com/site/ansisliit/Home/MyData.dat";
4. RemoteFile rFile=new ProxyRemoteFile(location);
5. rFile.viewFile();
6. //view same file again
7. System.out.println("\nView same file again\n");
8. rFile.viewFile();
9. }
10. }

```

output:

```

Loading Remote Resource
Displaying the remote file...

* This is the remote file/resource *
* http://javaxclass.blogspot.com *
* Proud to be a Sri Lankan! *

Time taken to view : 0.038747151109999996 s

View same file again

Displaying the remote file...

* This is the remote file/resource *
* http://javaxclass.blogspot.com *
* Proud to be a Sri Lankan! *

Time taken to view : 1.36416E-6 s

```

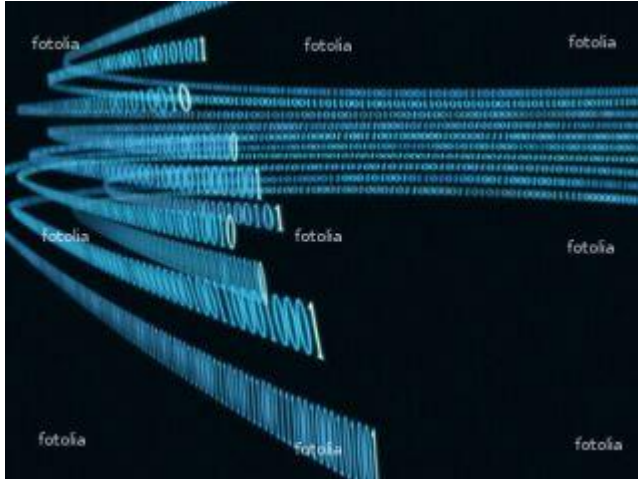
පැහැදිලි කිරීම;

මුල් වතාවේ viewFile() method එක call කල විට එය view වීම සඳහා තත්පර 0.0387 ක් ගතවී ඇති බවත් නැවත වතාවක් එකම RemoteFile object එක සඳහා එම viewFile() method එක call කල විට ගතවී ඇත්තේ ඉතාමත්ම සුළු කාලයක් එනම් තත්පර  $1.3641 \times 10^{-6}$  ක් බවත් ඔබට පෙනෙනවා ඇති.

අප විසින් සාර්ථකව Proxy design pattern එක මෙම ගැටලුවට ආදේශ කල නිසා එකම resource එක නැවත නැවතත් ප්‍රවේශනය වීම වැළැක්වී තිබේ.

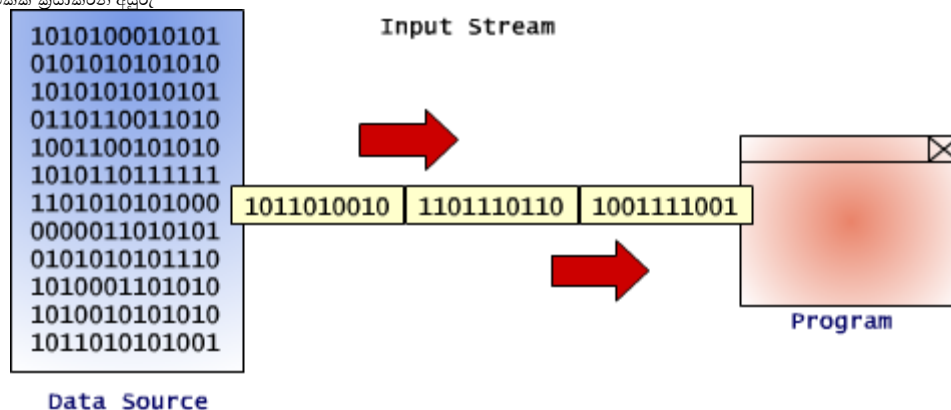
# ජාවා තුළ යෙදෙන I/O Streams යනු මොනවාද | Java I/O Streams

Tuesday, December 7, 2010 by Kaniskha Dilshan



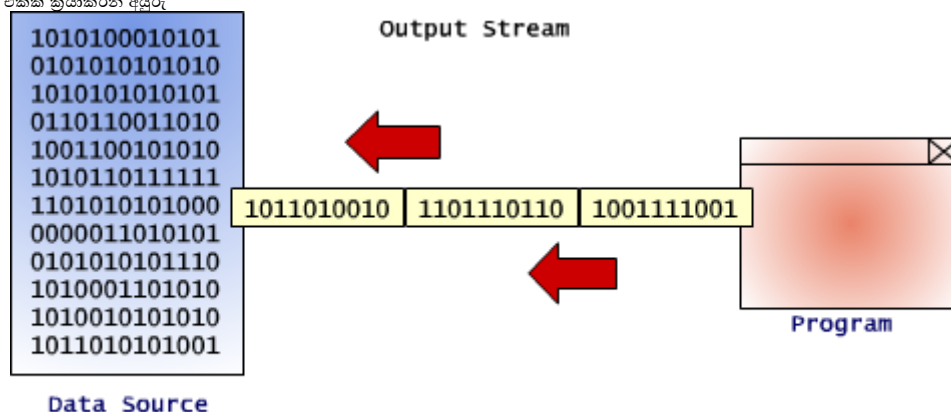
1000100010000000000100

පරිගණක වැඩසටහන් මගින් Data Source එකක් ප්‍රවේශය කිරීමේදී(Access) සිදුවන ක්‍රියාවලිය සහන පරිදි රූපමය වශයෙන් පෙන්විය හැකිය.  
Input Stream එකක් ක්‍රියාකරන අයුරු



(c) <http://javaxclass.blogspot.com>

Output Stream එකක් ක්‍රියාකරන අයුරු



(c) <http://javaxclass.blogspot.com>

\*කහ පාටින් දැක්වෙන්නේ Stream එකයි.

ඇතැම් ස්ට්‍රීම් එකලිනි දත්ත ගමන් කරවීමට සහය දක්වන අතර(උදා : [FileInputStream](#).) ඇතැම් ඒවා එකලිනි යන දත්ත වෙනත් ආකාරයකට/ස්වභාවයකට පත්කරනු ලබයි(උදා: [GZIPInputStream](#), [DeflaterOutputStream](#)).

Streams තුළින් යටත්තාවූ දත්ත වල ස්වභාවය අනුව මූලික වශයෙන් ස්ථිති වර්ග 2ක් හඳුනාගත හැකිය.

- Character Streams

මෙමගින් මිනිසාට කියවිය හැකි ආකාරයේ දත්ත ගලා යාම සිදුකෙරේ(උදා : ඉලක්කම්, අකුරු යනාදිය)

- Byte Streams

මේවායේ යටතුවේ machine-formatted ආකාරයේ දත්තවේ. මේවා පවතිනුයේ ද්විමය(binary) ආකාරයෙනි(උදා:110110). අදාලතාවය අනුව සුදුසු Stream ආකාරය තෝරාගත යුතුය. අනෙකුත් සියලුම ස්ත්‍රීම් වර්ගයන්හි පදනම වනුයේ Byte Stream වේ.

සුදුසු Stream එක තෝරාගන්නේ කෙසේද?

- කිසියම් data source එකක් low-level මට්ටමෙන් ප්‍රවේශනය කිරීමට අවශ්‍ය විට byte stream එක සුදුසුය.උදා: data file copy කිරීම, Audio files edit කිරීම යනාදියේදී.
- Data source එක text data වලින් සමන්විත නම් සහ text manipulations අවශ්‍ය විටදී character streams භාවිතාකල හැකිය.

ජාවා වැඩසටහනක් තුල මෙම ස්ත්‍රීම් භාවිතා වන්නේ කේසේදැයි අපි දැන් බලමු.  
මුලින්ම බලමු character stream කියවීමට FileReader යොදාගන්නා අයුරු.

```
/**
 *class : CharOperations
 *Author : Kanishka Dilshan
 *Purpose: Showing how to use character streams
 *Blog : http://javaxclass.blogspot.com
 */

import java.io.*;

public class CharOperations{
 public static void main(String args[]){
 FileReader fileRdr=null;
 FileWriter fileWr=null;
 try{
 fileRdr=new FileReader("myCharFile.txt");
 fileWr=new FileWriter("newCharFile.txt");
 int tmp;
 while((tmp=fileRdr.read())!=-1){
 fileWr.write(tmp);
 }
 }catch(FileNotFoundException e){
 System.err.println("File Not Found : " + e.getMessage());
 }catch(IOException e){
 System.err.println("IO Exception : " + e.getMessage());
 }finally {
 try{
 if(fileRdr!=null){
 fileRdr.close();
 }
 }catch(IOException e){}
 try{
 if(fileWr!=null){
 fileWr.close();
 }
 }catch(IOException e){}
 }
 }
}
```

පැහැදිලි කිරීම:

FileReader සහ FileWriter යනු පිළිවෙලින් myCharFile.txt සහ newCharFile.txt ට ලිවීමට තෝරාගත් Stream classes වේ. ඒවායින් සාදාගත් fileRdr සහ fileWr objects යොදාගෙන source file එකෙන් කියවීම සහ destination file එකට ලිවීම සිදුකෙරේ. මේවා character streams නිසා කියවීම සහ ලිවීම සිදුකරනුයේ character ආකාරයට ය. while loop එක තුලදී සිදුකෙරෙනුයේ input stream එකෙන් character by character කියවා output stream එකට character by character ලිවීමයි.

එහිදී while loop එකේ condition එක වශයෙන් යොදා ඇත්තේ කියවන ලද character එකෙහි integer අගය -1 ට සමානදැයි බැලීමයි. එම කියවන ලද අගය -1 නම් එයින් සදහන් වනුයේ අප සිටින්නේ EOF(End of the file) හි බවය(උදා: එකෙහි අවසානයේ බවය). මේවා try block එකක් තුල යෙදීමට හේතුව ස්ත්‍රීම් සමග ගනුදෙනු කිරීමේදී පැනනැගිය හැකි exception හසුකර ගැනීමයි.(Exception පිළිබඳ මීට පෙර සාකච්ඡා කර තිබේ). අවසානයේ open කරගන්නා ලද ස්ත්‍රීම් වසාදැමීම සිදුකර ඇත.( line 29 සහ line 34) මෙසේ ස්ත්‍රීම් close කිරීම අනිවාර්යයෙන්ම කල යුත්තකි. නැතහොත් අප ලියන වැඩසටහන් වල resource leaks ඇතිවීමට හේතුවේ. එය වැඩසටහනක ඇති විශාල ගැටලුවකි.

ලිපිය දීර්ඝ වන නිසා Byte streams පිළිබඳ වැඩිදුර තොරතුරු අපි ඊළඟ පාඩමෙන් කතාකරමු.

# ජාවා Byte Streams භාවිතා කරන අයුරු

Wednesday, December 8, 2010 by Kaniskha Dilshan

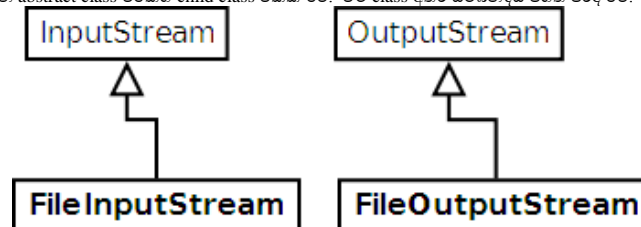
කලින් පාඩමෙන් අපි කතාකලා ජාවා stream පිළිබඳව. දැන් අපි බලමු byte stream භාවිතා කරන්නේ කුමනාකාරයෙන්ද කියලා. Byte stream එකක් input / output operations සිදු කරනුයේ 8-bit අකාරයටයි. එනම් character streams වලදී මෙන් නොව මෙහිදී characters වෙනුවට raw bytes කියවීම සහ ලිවීම සිදුකෙරේ. එම නිසා data file copy කිරීම යනාදී වැඩ වලට යොදාගත යුත්තේ byte stream වේ. නමුත් භාවිතාවන data, character වලින් යුක්ත නම් character streams වලට යෑම වඩාත් ඵලදායීවේ.

අහන දැක්වෙන ආකාරයට ජාවා වැඩසටහනක් කුලදී byte streams යොදාගත හැකිය. මෙහිදී I/O Stream නිරූපණය කිරීම සඳහා java.io පැකේජයේ ඇති FileInputStream සහ FileOutputStream class යොදාගැනේ.

```
1. /**
2. *class : ByteOperations
3. *Author : Kanishka Dilshan
4. *Purpose: Showing how to use byte streams
5. *Blog : http://javaclass.blogspot.com
6. */
7.
8.
9. import java.io.*;
10.
11. public class ByteOperations{
12. public static void main(String args[]){
13. FileInputStream fileInpStrm=null;
14. FileOutputStream fileOutStrm=null;
15. try{
16. fileInpStrm=new FileInputStream("myDataFile.dat");
17. fileOutStrm=new FileOutputStream("newDataFile.dat");
18. int tmp;
19. while((tmp=fileInpStrm.read())!=-1){
20. fileOutStrm.write(tmp);
21. }
22.
23. }catch(FileNotFoundException e){
24. System.err.println("File NOT Found : " + e.getMessage());
25. }catch(IOException e){
26. System.err.println("IO Exception : " + e.getMessage());
27. }finally {
28. try{
29. if(fileInpStrm!=null){
30. fileInpStrm.close();
31. }
32. }catch(IOException e){}
33. try{
34. if(fileOutStrm!=null){
35. fileOutStrm.close();
36. }
37. }catch(IOException e){}
38. }
39. }
40. }
```

මෙහිදී සිදුවනුයේ myDataFile.dat file එකේ clone එකක් newDataFile.dat ලෙසින් සෑදීමයි. [පෙර](#) පරිදිම මෙහිදීද open කරන ලද ස්ත්‍රීම close කර තිබේ.

FileInputStream යනු InputStream යන abstract class එකෙහි child class එකක් වේ. ඒම class අතර සම්බන්දය පහත පරිදි වේ.

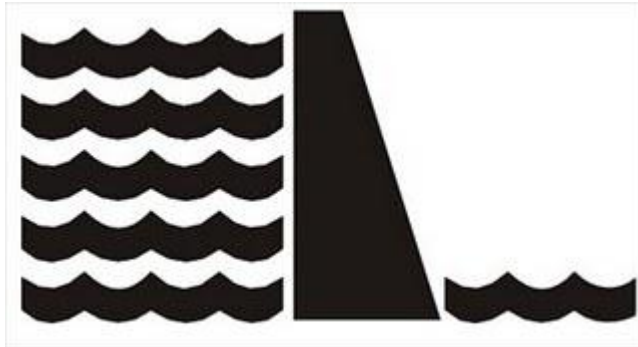


තවත් ස්ත්‍රීම වර්ග හා ඒවා යොදාගන්නා ආකරයන් පිළිබඳව අපි ඉදිරි පාඩම් වලදී සාකච්ඡා කරමු.



# ජාවා වැඩසටහන් තුළ Buffered Stream යොදාගන්නා අයුරු

Thursday, December 16, 2010 by Kaniskha Dilshan



මීට පෙර පාඩම් වලදී අපි Stream පිළිබඳ කතාකලානෙ අපි මෙම පාඩමෙන් තවත් වැදගත් Stream වර්ගයක් වන Buffered Stream පිළිබඳ අධ්‍යයනය කරමු.

අපි මීට පෙර සලකා බලන ලද FileInputStream , FileReader යන Stream වර්ග සැලකූ විට කෙලින්ම ඒවා යොදාගෙන කිසියම් resource එකක් භාවිතා කිරීමේදී ඒවායෙන් ලබාදෙන සැම read, write request එකක් සඳහාම අදාළ resource එක access කිරීම සිදුවනවා. උදාහරණයක් ලෙසට අපි හිතමු අපිට access කිරීමට අවශ්‍ය resource එක disk file එකක් කියලා. එහිදී සැම read, write request එකකදීම hard disk එකේ අදාළ file එක access කිරීම සිදුවනවා. මෙහිදී JVM (Java Virtual Machine) එක විසින් native calls(operating system එකට අදාළ low level methods) භාවිතා කිරීම සිදුකෙරෙනවා. එනම් සැම read, write request එකකටම native calls invoke කිරීමට සිදුවනවා. තවත් උදාහරණයක් ලෙස අපි සලකමු මෙම resource එක client Socket එකක් කියලා. එවිට ඉහත ආකාරයේ buffered වැටහෙනවා ඇති Buffered Stream භාවිතයේදී ඉහත සඳහන් කළ Buffered නොවන stream වලදී ඇතිවන ගැටලුවට සාර්ථක විසඳුමක් ලැබී තිබෙන ආකාරය.

නොවන Stream එකක් යොදාගන්නා සැම read, write request එකක් සඳහාම network resources භාවිතාවීම සිදුවනවා. කායික්ෂමතාව අතින් සැලකීමේදී මෙය හොඳ විසඳුමක් වන්නේ නැහැ.

අපි දැන් Buffered Stream එකකදී සිදුවන ක්‍රියාවලිය කුමක්දැයි සලකා බලමු, Buffered Stream එකකදී read, write යනාදිය සිදුකරනුයේ සෘජුව අදාළ resource එක සම්බන්ධී නොවේ "buffer" නම් memory area එකකිනි. මේ සඳහා "buffer" යනුවෙන් හැඳින්වෙන ප්‍රදේශයක් ප්‍රධාන මතකයේ (main memory/RAM) වෙන්ව පවතී. buffer එක හිස්වූ විට පමණක් OS එකට අදාළ low level methods invoke වීම සිදුවී "buffer" එක පිරේ(read කරන විටදී). RAM එක සම්බන්ධ I/O operations දාඩ තැවිය සම්බන්ධ I/O operations වලට වඩා ඉතා වේගවත් නිසා Buffered stream යොදාගෙන I/O operations සිදුකිරීමේදී උපරිම කාර්යක්ෂමතාවයක් ලබාගත හැකිය. දැන් ඔබට වැටහෙනවා ඇති Buffered Stream භාවිතයේදී ඉහත සඳහන් කළ Buffered නොවන stream වලදී ඇතිවන ගැටලුවට සාර්ථක විසඳුමක් ලැබී තිබෙන ආකාරය.

Buffered Stream එකක් සකසාගන්නේ කොහොමද?

මෙහිදී සිදුකරන්නේ non buffered stream object එකක් buffered stream object එකක් මගින් wrap කිරීමයි. පහත දැක්වෙන පරිදි මෙය සිදුකළ හැකිය.

Syntax 01

1. //for input operations
2. FileReader fileRdr=new FileReader("myFile.txt");
3. BufferedReader buffRdr=new BufferedReader(fileRdr);
- 4.
5. //for output operations
6. FileWriter fileWtr=new FileWriter("myFile2.txt");
7. BufferedWriter buffWtr=new BufferedWriter(fileWtr);

Syntax 02 (ඉහත ක්‍රියාවලියම කෙටියෙන්)

1. //for input operations
2. BufferedReader buffRdr=new BufferedReader(new FileReader("myFile.txt"));
- 3.
4. //for output operations
5. BufferedWriter buffWtr=new BufferedWriter(new FileWriter("myFile2.txt"));

ඉහත උදාහරණ වලින් පෙන්වා ඇත්තේ character stream handle කිරීම සඳහා buffered stream යොදාගන්නා ආකාරයයි. Byte streams සඳහා [BufferedInputStream](#) සහ [BufferedOutputStream](#) යොදාගත හැකිය.

Buffered stream එකක buffer size එක ලබාදෙන්නේ කොහොමද?

අපි ඉහත දක්වා ඇති උදාහරණ වලදී buffer size එකක් ලබාදී නොමැත constructor එකෙහි ලබාදී ඇත්තේ wrap කිරීමට stream object එක පමණි. එවිට JVM එක විසින් කිසියම් default අගයක් buffer size එක ලෙස තෝරා ගනී. මෙම අගය රඳාපවතිනුයේ OS එක, system configuration එක, Main memory ධාරිතාවය වැනි සාධක මතය. නමුත් අපට buffer size එක අපේ අභිමතය පරිදි ලබාදිය යුතු නම් buffered input සහ buffered output යන stream දෙවර්ගයටම buffer size ලබාදිය හැකිය. පහත දැක්වෙන අන්දමට එය සිදුකළ හැකිය.

Syntax 03

1. BufferedReader bfrRdr=new BufferedReader(new FileReader("myFile.txt",512\*1024));
2. BufferedWriter bfrWtr=new BufferedWriter(new FileWriter("myFile2.txt",512\*1024));

ඉහත උදාහරණයේදී Buffer size එක ලෙස 512KB ප්‍රමාණයක් ලබාදී ඇත මෙම අගය bytes වලින් ලබාදිය යුතු නිසා (512\*1024) ලෙස ලබාදී ඇත.

Buffered Stream එකක් flush කිරීම යනු කුමක්ද?

ඇතැම් වැදගත් අවස්ථා වලදී buffer එකේ ඇති තොරතුරු resource එකට ලිවීමට සිදුවන අවස්ථා පැමිණේ. මෙවැනි අවස්ථාවකදී flush නම් ක්‍රියාවලිය සිදුකරයි. මේ සඳහා flush() නම් විධානය භාවිතා කරයි.

Non buffered stream සඳහා flush() method එක පවතී. නමුත් ඒවායේදී flush() කිරීම මගින් කිසිම ප්‍රතිඵලයක් නොමැත. එයට හේතුව ඔබට මේ වනවිට සිතාගත හැක.

Non buffered stream වලදී flush කිරීමට memory area එකක් නොමැති වීම මීට හේතුවයි.

Java 1.6 Documentation එකෙහි දැක්වෙන පරිදි Buffered Stream සඳහා අදාළ වන constructors සහ methods පහත දක්වා ඇත.

## Class `BufferedInputStream`

### Constructor Summary

|                                                                           |                                                                                                                                        |
|---------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| <code>BufferedInputStream</code> ( <code>InputStream</code> in)           | Creates a <code>BufferedInputStream</code> and saves its argument, the input stream in, for later use.                                 |
| <code>BufferedInputStream</code> ( <code>InputStream</code> in, int size) | Creates a <code>BufferedInputStream</code> with the specified buffer size, and saves its argument, the input stream in, for later use. |

### Method Summary

|         |                                                                                                                                                                                                             |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| int     | <code>available()</code><br>Returns an estimate of the number of bytes that can be read (or skipped over) from this input stream without blocking by the next invocation of a method for this input stream. |
| void    | <code>close()</code><br>Closes this input stream and releases any system resources associated with the stream.                                                                                              |
| void    | <code>mark</code> (int readlimit)<br>See the general contract of the <code>mark</code> method of <code>InputStream</code> .                                                                                 |
| boolean | <code>markSupported()</code><br>Tests if this input stream supports the <code>mark</code> and <code>reset</code> methods.                                                                                   |
| int     | <code>read()</code><br>See the general contract of the <code>read</code> method of <code>InputStream</code> .                                                                                               |
| int     | <code>read</code> (byte[] b, int off, int len)<br>Reads bytes from this byte-input stream into the specified byte array, starting at the given offset.                                                      |
| void    | <code>reset()</code><br>See the general contract of the <code>reset</code> method of <code>InputStream</code> .                                                                                             |
| long    | <code>skip</code> (long n)<br>See the general contract of the <code>skip</code> method of <code>InputStream</code> .                                                                                        |

## Class `BufferedOutputStream`

### Constructor Summary

|                                                                              |                                                                                                                              |
|------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------|
| <code>BufferedOutputStream</code> ( <code>OutputStream</code> out)           | Creates a new buffered output stream to write data to the specified underlying output stream.                                |
| <code>BufferedOutputStream</code> ( <code>OutputStream</code> out, int size) | Creates a new buffered output stream to write data to the specified underlying output stream with the specified buffer size. |

### Method Summary

|      |                                                                                                                                                          |
|------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| void | <code>flush()</code><br>Flushes this buffered output stream.                                                                                             |
| void | <code>write</code> (byte[] b, int off, int len)<br>Writes len bytes from the specified byte array starting at offset off to this buffered output stream. |
| void | <code>write</code> (int b)<br>Writes the specified byte to this buffered output stream.                                                                  |

ඉහත දැක්වෙන methods භාවිතාකරන අන්දම හොඳින් වටහාගැනීමෙන් වඩාත් කාර්යක්ෂම වැඩසටහන් ලිවිය හැකිය.  
අපි මීලඟ පාඩමෙන් Buffered Stream යොදාගෙන ප්‍රායෝගික ජාවා වැඩසටහනක් ලියන අන්දම බලමු.

# Buffered Stream යොදාගෙන File Copy කරන අන්දම

Friday, December 17, 2010 by Kanishka Dilshan



අපි මීට පෙර පාඩමෙන් Buffered stream පිළිබඳව අධ්‍යයනය කලා. අපි මෙම පාඩමෙන් එම සිද්ධාන්ත යොදාගනිමින් File copy කිරීම සඳහා වැඩසටහනක් ලිවීමට උත්සාහ කරමු. මේ සඳහා අපි එම buffered stream classes තුළ ඇති read() , write() , flush() , close() යන methods භාවිතයට ගනිමු. පහත දැක්වෙන්නේ ඒ සඳහා වන කේත කොටසයි.

**FileOperations.java**

```
1. /**
2. *class : FileOperations
3. *Author : Kanishka Dilshan
4. *Purpose: Using buffered Streams to copy a file
5. *Blog : http://javaxclass.blogspot.com
6. */
7.
8. import java.io.*;
9.
10. class FileOperations {
11. public static final int BUFF_SIZE=8*1024*1024;//8 MB
12.
13. public boolean fileCopy(String source,String dest){
14. boolean isCopied=false;
15. BufferedInputStream buffIn=null;
16. BufferedOutputStream buffOut=null;
17. byte buffer[]=new byte[BUFF_SIZE];
18. try{
19. buffIn=new BufferedInputStream(new FileInputStream(source),8*1024*1024);
20. buffOut=new BufferedOutputStream(new FileOutputStream(dest),8*1024*1024);
21.
22. int len;
23. while((len=buffIn.read(buffer))>0){
24. buffOut.write(buffer,0,len);
25. buffOut.flush();
26. }
27. isCopied=true; //file copied successfully
28. }catch(FileNotFoundException e){
29. System.err.println("Cannot find the file : " + e.getMessage());
30. }catch(IOException e){
31. System.err.println("IO Error : " + e.getMessage());
32. }finally{
33. if(buffIn!=null){
34. try{
35. buffIn.close();
36. }catch(IOException e){}
37. }
38. if(buffOut!=null){
39. try{
40. buffOut.close();
41. }catch(IOException e){}
42. }
43. }
44. return isCopied;
45. }
46. }
```

**FileCopy.java**

```
1. class FileCopy{
2. public static void main(String args[]){
3. if(args.length!=2){
4. System.out.println("Incorrect syntax!");
5. System.out.println("Use Following Syntax");
6. }
7. }
8. }
```

```

6. System.out.println("java FileCopy #source_file# #estination_file#");
7. System.exit(0);
8. }
9.
10. FileOperations fo=new FileOperations();
11. boolean status=fo.fileCopy(args[0],args[1]);
12. if(status==true){
13. System.out.println("File was copied successfully!");
14. }else{
15. System.out.println("File was not copied");
16. }
17.
18. }
19. }

```

අපි දැන් ඉහත වැඩසටහනේ ක්‍රියාකාරීත්වය අධ්‍යයනය කරමු.

**Line 11 :** මෙහිදී BUFF\_SIZE ලෙස 8MB constant අගයක් declare කර ගැනීම සිදුකෙරෙනවා. පසුව buffer එකේ ඇතිදැ කියවීමට සහ buffer එකට ලිවීමට මෙම constant අගයට අනුව සකසන byte array එකක් යොදාගනු ලබනවා.

**Line 12:** boolean isCopied=false; මෙය යොදාගන්නේ file එක සාර්ථකව කොපි වුවාද නැද්ද යන්න return කිරීම සඳහායි.

**Line 17:** byte buffer[]=new byte[BUFF\_SIZE]; මෙහිදී 8MB ප්‍රමාණයකින් යුතු "buffer" නම් byte array එකක් සකසාගනු ලැබේ.

**Line 19, 20 :** buffIn සහ buffOut යනුවෙන් buffered stream object 2ක් සකසාගනු ලැබේ(class type සලකා බලන්න) මින් buffIn object එක යොදාගනුයේ source file එක කියවීමටය. එමෙන්ම buffOut යොදාගනුයේ destination file එකට ලිවීමටය. මෙම object සැදීමේදී ඒවාට අදාල buffer size එකද විශේෂයෙන් සඳහන් කර ඇත.

**Line 23 :** මෙම while loop එක තුළදී මූලිකවම කාඨයන් 2 ක් සිදුකෙරේ len=buffIn.read(buffer) මගින් සිදුකරන්නේ input stream එකෙන් කියවන ලද අගයන් buffer නම් byte array එකේ store කර len නම් int විචල්‍යය තුළ කියවන ලද දේවා ප්‍රමාණය assign කිරීමයි.

එහිදී සිදුකරන අනෙකුත් කාඨය නම් එම කියවන ලද දේවා ප්‍රමාණය 0 ට වඩා වැඩිද යන්න පරීක්ෂා කර බැලීමයි. කියවන ලද දේවා ප්‍රමාණය -1 ලෙස ලැබේ නම් ඉන් ගමා වනුයේ stream එකෙහි අවසානයට ලඟාවී ඇති බවයි නැතහොත් end of file(EOF) බවයි මෙවිට output stream එකට ලිවීමට අවශ්‍ය නැත. කියවන ලද දේවා ප්‍රමාණය 0 නම්ද output stream එකට ලිවීමට අවශ්‍ය නැත. එම නිසා (len=buffIn.read(buffer))>0 නම් condition එක true නම් පමණක් while loop එක ක්‍රියාත්මකවේ.

**Line 24 :** buffOut.write(buffer,0,len); මෙහිදී සිදුකරනුයේ කියවන ලද දේවා (byte array එක) output stream එකට ලිවීමයි. 0 සහ len යන parameters වලින් කියවෙනුයේ එම byte array එකේ 0 සිට len යන අගය දක්වා byte ප්‍රමාණය output stream එකට ලිවිය යුතු බවයි.

**Line 25 :** buffOut.flush(); මෙහිදී output stream එක clear කිරීම සිදුකරයි.

**Line 27 :** while loop එක අවසන් වී මෙම line එකට පැමිණියා යනු file එක සාර්ථකව ලියා අවසන් බවට සහතික විය හැකිය මන්දයත් IO Operations සිදුකර ඇත්තේ exception handle කල හැකි පරිදි try-catch block තුළය. එම නිසා මෙහිදී isCopied නම් විචල්‍යයේ අගය true ලෙස සටහන් කරනු ලැබේ.

**Line 32:** catch block එක තුළදී සිදුකර ඇති දෑ පහසුවෙන් තේරුම් ගත හැකි නිසා finally block තුළදී කුමක් සිදුවන්නේද යන්න සලකා බලමු. මෙහිදී ඉහත open කරගන්නා ලද input සහ output stream 2ම close කිරීම සිදුකරයි මෙය සිදුකරන්නේ තවත් try-catch block එකක් තුළදීය.

වැඩසටහන මගින් source file එකේ binary pattern එක ඒ ආකාරයෙන්ම destination file එකේ තිබේ දැයි බැලීමට [hash calculator](#) එකක් යොදාගත හැක.

මෙම ක්‍රියාදාමය වටහාගැනීම සඳහා ඉහත පැහැදිලි කිරීම ප්‍රමාණවත් යයි සිතමි. කෙසේ වෙතත් තවදුරටත් පැහැදිලි කල යුතු යමක් ඇත්නම් ඉදිරිපත් කරන මෙන් ඉල්ලා සිටින්න කැමතියි.

# ජාවා Data Streams භාවිතා කරන අයුරු

Saturday, December 18, 2010 by Kaniskha Dilshan



දැනගැනීමට පුළුවන්.

කලින් අධ්‍යයනය කළ stream වලදී මෙන් නොව data stream වලදී File එකේ අවසානය (EOF) හඳුනාගැනීමට සෘජු ක්‍රමයක් නැත. එම නිසා EOFException එක catch කිරීමෙන් file එකක අවසානය හඳුනාගතයුතු වනවා.

DataOutputStream භාවිතාකරන අන්දම...

මෙහිදී writeInt(), writeLong(), writeUTF()... යනාදී වශයෙන් primitive data ලිවීමට වැදගත් වන methods ගතයාවක් තිබෙනවා. ඊට පෙර stream වලට පොදුවූ flush(), close() යනාදී methods ද මෙම data stream විසින් inherit කරගෙන සිටිනවා.

අපි දැන් බලමු data stream එකක ක්‍රියාකාරීත්වය වටහාගැනීමට පහසුවන අන්දමේ සරල ජාවා වැඩසටහනක්.

```
1. /**
2. *class : Test
3. *Author : Kanishka Dilshan
4. *Purpose: Demonstrate how to use data streams
5. *Blog : http://javaxclass.blogspot.com
6. */
7.
8. import java.io.*;
9.
10. class Test {
11. public static void main(String args[]){
12. try{
13. DataOutputStream dout=new DataOutputStream(
14. new BufferedOutputStream(
15. new FileOutputStream("myDataFile.dat")));
16.
17. dout.writeUTF("Kanishka");
18. dout.writeFloat(83.23f);
19. dout.writeBoolean(true);
20.
21. dout.writeUTF("Ravindu");
22. dout.writeFloat(98.12f);
23. dout.writeBoolean(false);
24.
25. dout.writeUTF("Madhawa");
26. dout.writeFloat(84.65f);
27. dout.writeBoolean(false);
28.
29. dout.close();
30.
31. DataInputStream din=new DataInputStream(
32. new BufferedInputStream(
33. new FileInputStream("myDatafile.dat")));
34. try{
35. while(true){
36. String name=din.readUTF();
37. float marks=din.readFloat();
38. boolean val=din.readBoolean();
39. System.out.println(name+" "+marks+" "+val);
40. }
41. }catch EOFException e){
42. System.out.println("End of the data file");
```

මේ වනවිට අපි ඉතා වැදගත් stream වර්ග කිහිපයක් ගැන කතාකලා. byte stream, character stream, buffered stream එම stream පිළිබඳ ඔබට මේ වනවිට හොඳ අවබෝධයක් ඇති බව උපකල්පනය කරමින් අපි මෙම පාඩමෙන් සාකච්ඡා කිරීමට යන්නේ data streams පිළිබඳවයි.

Data stream බාවිතයෙන් stream එකකට primitive data ලිවීමට සහ data stream එකකින් primitive data කියවීමට පුළුවන්. එනම් int, float, double, long, boolean, char, byte මෙන්ම primitive data type එකක් නොවුවද String සඳහා data streams සහය දක්වනවා.

[DataInputStream](#) සහ [DataOutputStream](#) නම් class වර්ග 2 යොදාගෙන data streams නිරූපණය කෙරෙනවා. එවන් stream එකකින් දත්ත කියවීමට DataInputStream ද එව්ට ලිවීමට DataOutputStream class එකද යොදාගැනෙනවා. හොඳින් මතක තබාගත යුතු කාරණාවක් තමයි සෑම විටම data stream object එකක් construct කළ යුත්තේ byte stream එකක් wrap කිරීමෙන් බව.

DataInputStream භාවිතාකරන අන්දම...

මෙහිදී readInt(), readLong(), readUTF()... යනාදී වශයෙන් primitive data කියවීමට වැදගත් methods ගණනාවක් යොදාගන්න පුළුවන්. Java Documentation එක බැලීමෙන් මෙම methods සවිස්තරාත්මකව

```

43. }finally{
44. try{
45.
46. }catch(Exception e){
47. din.close();
48. }
49. }
50. }catch(Exception e){
51. System.err.println(e.getMessage());
52. }
53. }
54. }

```

Output :

```

C:\Documents and Settings\Kanishka
Kanishka 83.23 true
Ravindu 98.12 false
Madhawa 84.65 false
End of the data file

```

ඉහත code එක මගින් සිදුවන ක්‍රියාවලිය වටහා ගැනීමට උත්සාහ කරමු.

**Line 13,14,15** මගින් සුදුසු byte stream එකක් output data stream එකක් මගින් wrap කිරීම සිදුකර ඇත.

**Line 17,18,19** මෙහිදී කිසියම් පිළිවෙලකට අනුව data stream එකට දත්ත ලිවීම සිදුකර ඇත. එනම් මුලින් String data ද දෙවනුව float data ද අවසාන boolean data ද stream එකට එනම් myDataFile.dat file එකට ලියා ඇත.

**Line 31,31,33** මෙහිදී byte stream එකක් input data stream එකක් මගින් wrap කිරීම සිදුකර ඇත. මෙහිදී සාදාගන්නා din object එක හරහා data කියවීම සිදුකෙරේ.

**Line 34 - Line 48** මෙහිදී අදාළ data file එක අවසන් වනතෙක් කියවීම සිදුකර එම දත්ත console එකෙහි දර්ශනය කරයි. line 35 න් ආරම්භ වන while loop එක infinite loop එකක් ලෙස පෙනුනද file එකේ අවසානයේ EOFException එකක් throw කරන නිසා file එක අවසානය වනතෙක් පමණක් loop එක ක්‍රියාත්මක වේ.

Streaming වල තවත් වැදගත් කොටසක් වන Standard streams පිළිබඳ අපි මීලඟ පාඩමෙන් සාකච්ඡා කරමු.

# ජාවා වැඩසටහනක් සහ PHP වැඩසටහනක් අතර දත්ත හුවමාරු කරන අන්දම

Wednesday, December 29, 2010 by Kanishka Dilshan



එක දිගටම ජාවා සම්බන්ධ මූලධර්ම සාකච්ඡා කරපු නිසා අපි මේ පාඩමෙන් ප්‍රායෝගික පැත්තට බර කරුණු ටිකක් අධ්‍යයනය කරමු. මේ වන විට පරිගණක වැඩසටහනක් යනු හුදෙක්ම එක් පරිගණකයකට සීමාවූ දෙයක් නොවන බව අපි කවුරුත් දන්නා දෙයක්නේ. බොහොමයක් පරිගණක වැඩසටහන් අන්තර්ජාලයට සම්බන්ධව පවතිනවා. උදාහරණ වීදියට internet messenger වැඩසටහන් , virus guards, වැනි වැඩසටහන් දක්වන්න පුලුවන්. වයිරස් ගාඩ එකක් සැලකුවොත් එය අන්තර්ජාලය හා සම්බන්ධ වන අවස්ථා තමයි අලුත් virus definition නිබේද්‍ය පරීක්ෂා කිරීම, අලුත් virus definition නිබේදනම් ඒවා ලබා ගැනීම, සැක සහිත ගොණුවක් අදාළ ආයතනයට upload කිරීම වැනි අවස්ථා. මෙහිදී මූලිකවම සිදුවන්නේ දුරස්ථ සර්වරයක ඇති වැඩසටහනක් හා කිසියම් සන්නිවේදනයක් කිරීමයි.

අපි මෙම පාඩමෙන් අධ්‍යයනය කිරීමට යන්නේ ජාවා වැඩසටහනකින් දුරස්ථ වැඩසටහනක් සමග සම්බන්ධ වන ආකාරයයි. මාතෘකාවට අනුව අපි දුරස්ථ වැඩසටහන ලෙස යොදාගන්නේ PHP script එකක්. නමුත් මෙම දුරස්ථ වැඩසටහන Python හෝ Perl script එකක් හෝ වෙනත් ඕනෑම server side script එකක් වීමට පුලුවන්.

කොහොමද දුරස්ථ වැඩසටහනට දත්ත යවන්නේ?

මේ සඳහා අපි යොදාගන්නේ POST method එකයි. HTTP (Hypertext Transfer Protocol) එකට අනුව දත්ත හුවමාරු කිරීමේදී GET සහ POST යනුවෙන් ක්‍රම දෙකක් පවතිනවා. ඉන් GET ක්‍රමයට අනුව දත්ත හුවමාරු කිරීමේදී අදාළ දත්ත URL එකට අමුණා යැවීම සිදුවනවා එනම් මෙම දත්ත පිටතට දෘෂ්‍යමාන වනවා එම නිසා password සම්බන්ධ දත්ත මෙම ක්‍රමයට හුවමාරු කරන්නේ නැහැ. අනික වරකට යැවිය හැකි දත්ත ප්‍රමාණයේ සීමාවක්ද පවතිනවා.

GET Request එකකට උදාහරණයක්

<http://mydomain.org/process.php?name=Kanishka&passw=hellox>

POST method එකේදී HTTP request එකේ header එක තුළ අදාළ දත්ත ඇතුළත්කොට යැවීම සිදුකරනවා. මෙම වැඩසටහනේදී අපි යොදාගන්නේ POST method එකයි. මෙම methods ගැන වැඩිදුර තොරතුරු දැනගැනීම සඳහා [මෙතනින්](#) පිවිසෙන්න.

අපිට මේ සඳහා මීට පෙර ඉගෙනගත් [streams](#) සම්බන්ධ දැනුම වැදගත් වනවා. මෙම වැඩ සටහනෙන් සිදුකරනුයේ දෙන ලද නගරයකට අදාළ area code එක ලබාදීමයි. මෙම PHP script එක පහත දැක්වෙනවා.

**telCodes.php**

```
1. <?php
2. /**
3. *Script : telCodes.php
4. *Author : Kanishka Dilshan
5. *Purpose : Processing a remote request
6. *Blog : http://javaclass.blogspot.com
7. */
8.
9. $code=$_POST['code'];
10. $file=fopen("codes.txt","r") or exit("Data file is missing!");
11. $status=FALSE;
12. while(!feof($file)){
13. $str=fgets($file);
14. if(strpos($str,$code)!=FALSE){
15. echo $str;
16. $status=TRUE;
17. break;
18. }
19. }
20. if($status===FALSE){
21. echo "code is not found!";
22. }
23. fclose($file);
24. ?>
```

මෙම වැඩසටහනේ 09 වන පේළියේදී POST request එක මගින් එවන ලද අදාළ දත්තය ලබාගන්නවා එය code නම් parameter එක සමග සම්බන්ධව පවතිනවා. ඉන්පසු data file එක විවෘත කර එවන ලද code එක එතුළ නිබේද්‍යව පේළියෙන් පේළිය කියවා පරීක්ෂා බලනවා. එය හමුවූ විට echo විධානය භාවිතාකර අදාළ response එක ලබාදෙනවා.

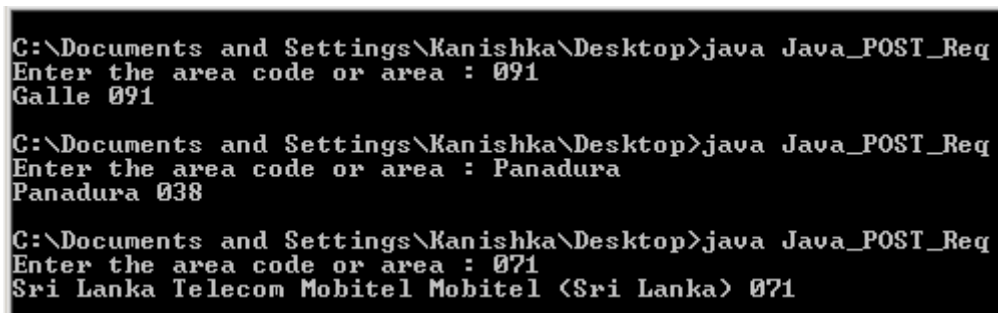


දැන් අපි බලමු මෙම PHP Script එකට දත්ත යවා එහි response එක ලබාගැනීමට ලිවිය හැකි ජාවා වැඩසටහන. මෙම ක්‍රියාවලිය සිදුකරගැනීමට අපි මෙහිදී යොදාගන්නේ net පැකේජය තුළ ඇතුළත් [URL](#) සහ [URLConnection](#) නම් classes දෙකයි. URLConnection එක streams කිහිපයකට සම්බන්ධ කරගෙන අදාළ ක්‍රියාවලිය සිදුකරගැනීම මෙහිදී මූලික වශයෙන් සිදුවනවා. මෙම ක්‍රියාවලිය Sockets යොදාගෙනද සිදු කිරීම කළ හැකියි. අපි ඉදිරි පාඩම පෙළකින් Sockets පිළිබඳ සාකච්ඡා කරමු.

**Java\_POST\_Req.java**

```
1. /**
2. *class : Java_POST_Req
3. *Author : Kanishka Dilshan
4. *Purpose: Making a post request to a php script and retrieve the response
5. *Blog : http://javaxclass.blogspot.com
6. */
7.
8. import java.net.*;
9. import java.util.Scanner;
10. import java.io.*;
11.
12. class Java_POST_Req{
13. public static void main(String args[]){
14. Scanner sc=new Scanner(System.in);
15. System.out.print("Enter the area code or area : ");
16. String value=sc.nextLine();
17.
18. OutputStreamWriter outWtr=null;
19. BufferedReader buffRdr=null;
20. try{
21. String httpdata=URLEncoder.encode("code","UTF-8")+"="+URLEncoder.encode(value,"UTF-8");
22. URL remoteResource=new URL("http://kanishka.hostei.com/telcodes/telCodes.php");
23. URLConnection conn=remoteResource.openConnection();
24. conn.setConnectTimeout(30*1000);
25. conn.setDoOutput(true);
26. outWtr=new OutputStreamWriter(conn.getOutputStream());
27. outWtr.write(httpdata);
28. outWtr.flush();
29.
30. //read the response
31. buffRdr=new BufferedReader(new InputStreamReader(conn.getInputStream()));
32. String line;
33. while((line=buffRdr.readLine())!= null){
34. System.out.println(line);
35. }
36. }catch(Exception e){
37. System.err.println("Error : "+e.getMessage());
38. } finally{
39. try{
40. outWtr.close();
41. buffRdr.close();
42. }catch(Exception e){ }
43. }
44. }
45. }
```

output :



```
C:\Documents and Settings\Kanishka\Desktop>java Java_POST_Req
Enter the area code or area : 091
Galle 091

C:\Documents and Settings\Kanishka\Desktop>java Java_POST_Req
Enter the area code or area : Panadura
Panadura 038

C:\Documents and Settings\Kanishka\Desktop>java Java_POST_Req
Enter the area code or area : 071
Sri Lanka Telecom Mobitel Mobitel <Sri Lanka> 071
```

මේ සඳහා ලියන ලද පහත දැක්වෙන applet එක භාවිතා කර මෙහි සිදුවන ක්‍රියාවලිය පිරික්සා බලන්න.

අපි දැන් මෙම ජාවා වැඩසටහන ජේෂ්‍රිය විස්තර කරගනිමු.

Line14 - Line 16 :

මෙහිදී Scanner object එකක් සාදාගෙන console එකෙන් user input එකක් ලබාගැනීම සිදුවනවා.

Line21:

මෙහිදී PHP වැඩසටහනට යැවීමට අවශ්‍ය data සූදානම් කිරීම සිදුකෙරෙනවා. encode("code","UTF-8") මගින් parameter එකද URLEncoder.encode(value,"UTF-8") මගින් අදාළ value එකද UTF-8 ක්‍රමයට එනම් unicode ක්‍රමයට encode කිරීම සිදුකෙරෙනවා.

Line22:

PHP script එකට අදාළ URL object එකක් සෑදීම මෙහිදී සිදුකෙරෙනවා.

Line23:

අදාළ PHP script එක සමග සම්බන්ධතාවය නිරූපණය වන URLConnection object එකක් මෙහිදී සකසාගැනීම සිදුකෙරෙනවා.



Line24:

connection timeout එක සඳහන් කිරීමෙන් සිදුකර ඇත්තේ අදාළ PHP වැඩසටහන සමග connection එකක් ස්ථාපනය කරගැනීමට උපරිම වශයෙන් තත්පර 30ක කාලයක් ලබාදීමයි.

Line25:

මෙමගින් URLConnection ඔබ්ජෙක්ට එක මගින් ඒ සමග සම්බන්ධ වී ඇති PHP වැඩසටහනට දත්ත ලිවීම සිදුකරන බව ඇඟවීම සිදුකරනවා.

Line26-Line27:

මෙහිදී නව OutputStreamWriter object එකක් සකසාගෙන එයට අදාළ URLConnection එකෙහි output stream එක සම්බන්ධ කරගෙන සකස් කරගත් HTTP data එම URLConnection එකෙහි output stream එකට ලියනු ලබනවා. එනම් අදාළ PHP script එක වෙත යවනු ලබනවා.

Line31-Line34:

PHP වැඩසටහන අප යවන HTTP request එක process කිරීමෙන් අනතුරුව ලබාදෙන response එක කියවීම මෙහිදී සිදුවනවා. මෙහිදී සකසාගෙන ඇති BufferedReader object එක සම්බන්ධ වී ඇත්තේ URLConnection එකෙහි input stream එකටයි. එනම් PHP වැඩසටහන මගින් ලබාදෙන output එකටයි.

Line36-Line42:

මෙ පිළිබඳව අපි Streams සම්බන්ධ පාඩම් වලදී විස්තරාත්මකව අධ්‍යයනය කර තිබෙන නිසා මෙහිදී එය නැවත විස්තර කිරීම අනවශ්‍යවේ.