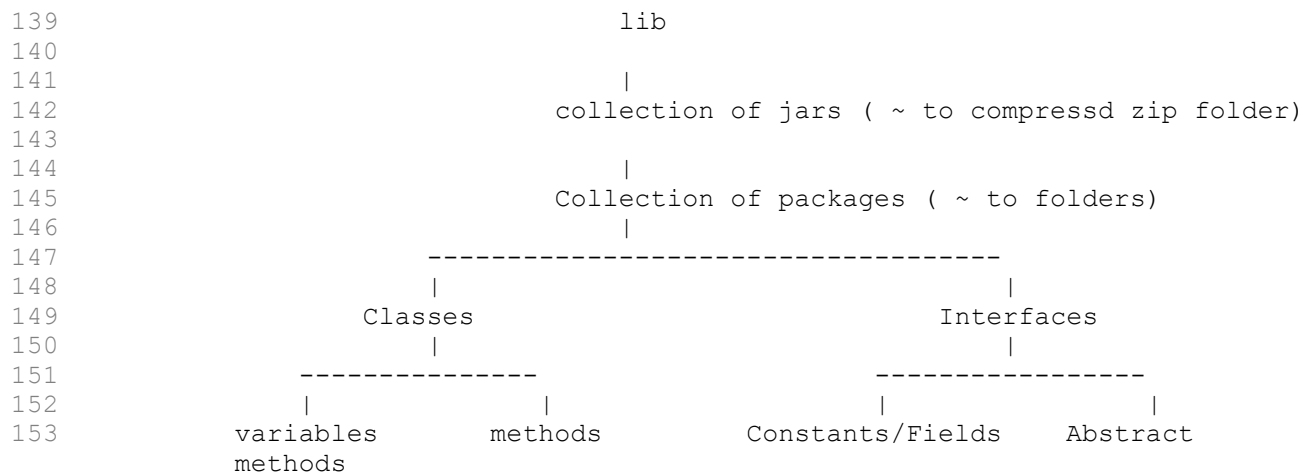


```
1
2
3 JAVA + Selenium
4
5
6 1996 - Sun Micro systems
7
8 now - Oracle
9
10
11 Features of Java :
12
13 1. Simple
14
15 - syntax is based/similar to c++
16
17 - removed many confusing or rarely used concepts/features like
18 explicit pointers , operator overloading
19
20 operators
21 + - * / %
22
23 2 + 5 = 7
24
25 + for power
26
27 2 + 5 = 32
28
29 - Automatic Garbage Collector
30
31 -----
32 2. OO(Object Oriented) Programming
33
34 POP Procedure /Functional Programming languages
35
36 OOP is a methodology that simplifies s/w development and maintenance
37 using some concepts/rules
38
39 - Object
40 - class
41 - Polymorphism
42 - Encapsulation
43 - Inheritance
44 - Abstraction
45
46 -----
47 3. Robust
48
49 - Java uses Strong memory management
50 - No pointers ==> avoid security problems
51 - Automatic Garbage Collector (unwanted memory is deallocated)
52 - Exception Handling and Type Checking Mechanism
53
54 int i=10;
55
56 String j="FLM";
57
58 int k=i+j; ==> error
59
60 -----
61
62 4. Secure :
63
64
65 -----
66 5. Platform Independent
67
68 OS
69
```

```
70 -----
71 6. Architectural Netural
72
73
74 Architecture = Micro Processor + RAM
75
76
77 -----
78 7. Portable
79
80 IBM
81
82 Apple
83
84 Solaris
85
86
87 -----
88
89 How to Install jdk :
90
91
92 jdk 8 or jdk 11 (is preferred)
93
94 download below file
95
96 jdk-11.0.20_windows-x64_bin.exe
97
98 to check jdk installed or not
99 go to below loaction
100
101 C:\Program Files\Java\jdk-11
102
103
104
105 Configuration of jdk :
106
107 Environment Variables
108
109 JAVA_HOME
110
111 C:\Program Files\Java\jdk-11
112
113 path
114
115 C:\Program Files\Java\jdk-11\bin
116
117 C:\Users\Riyaz>java -version
118 java version "11.0.20" 2023-07-18 LTS
119 Java(TM) SE Runtime Environment 18.9 (build 11.0.20+9-LTS-256)
120 Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.20+9-LTS-256, mixed mode)
121
122 -----
123
124
125
126 jdk
127
128
129
130
131 lib bin/tools jre
132
133
134 lib : Library
135
136 predefined stuff created by Java Communities
137
138
```



```

154
155
156
157
158 bin / tools :
159
160
161 to compile a java program

```

```

162 javac
163
164 syntax :
165
166 javac FileName.java

```

```

167
168
169 to excute a compiled java program :
170
171 java

```

```

172
173 syntax :
174
175 java FileName

```

```

176 -----

```

```

177
178 JRE : Java Runtime Environment

```

```

179
180
181             jvm (java virtual machine)

```

```

182
183
184
185
186             CLSS             EE             GC
187
188
189             Interpreter      JIT             Mark             Sweep
190             compiler

```

```

191
192 CLSS : Class Loader Subsystem

```

- ```

193
194 - Loading
195 - Linking
196 - Initialization

```

```

197
198
199
200 EE (Execution Engine):

```

```

201
202 Interpreter : BC to MC
203 byte code to machine code

```

```

204

```

```
207 and also execute the code line by line
208
209
210 JIT (Just In time) compiler :
211
212 looping ==> to give performance
213 for , while , etc ..
214
215
216 -----
217 Garbage Collector
218
219 for Memory deallocation
220
221 System.gc();
222 Runtime.gc();
223
224
225 -----
226 -----
227 JAVA Memory Blocks
228
229 1. Class Area / Method Area
230
231 - .class File (Programs)
232 - Static Variables
233 - Static Methods
234
235 2. Heap Area (dynamic memory allocation)
236
237 - Objects
238 - Arrays
239
240
241 3. Java Stack Area
242
243 - Local declarations / Local Variables
244 - expressions
245
246 int x=10,y=20;
247
248 x+y ==> 30
249
250 - current running logic /code
251
252 4. String constant pool Area (SCPA)
253
254 only string literals
255
256 String str="FLM";
257
258
259 -----
260 Structure of Java Programs
261
262 4 Sections
263
264 1. Documentation Section
265
266 2. Package Section
267
268 3. Import Section
269
270 4. Class/Interface Section
271
272
273 -----
274 1. Documentation Section
275 -----
```

```

276
277 comments
278
279 single line comment :
280
281 // kjhkhkhkh
282
283 multiline comment :
284
285 /* kjhkhkhkhkhkh
286
287 khkhkhkhkhkhkhkh
288
289 khkhkhkhkhkhkhkh */
290
291 -----
292 2. Package Section
293 -----
294
295 Pacakage is similar to folder
296
297 logical grouping
298
299
300 syntax :
301
302 package package-name;
303
304 -----
305 3. Import Section
306 -----
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344

```

```

graph TD
 A[Collection of packages (~ to folders) (java.util)] --> B[Classes (Scanner)]
 A --> C[Interfaces]
 B --> D[variables]
 B --> E[methods]
 C --> F[Constants/Fields]
 C --> G[Abstract methods]
 E --> H[nextInt()]

```

import packagename.\*; ==>not preferred  
import packagename.ClassName; gives more readability  
import static package-name.classname.\*;

java.lang is a default package in java

there is no need of import

```

4. Class/Interface Section

```

access-specifier access-modifier class ClassName  
{  
//variables  
//methods

```

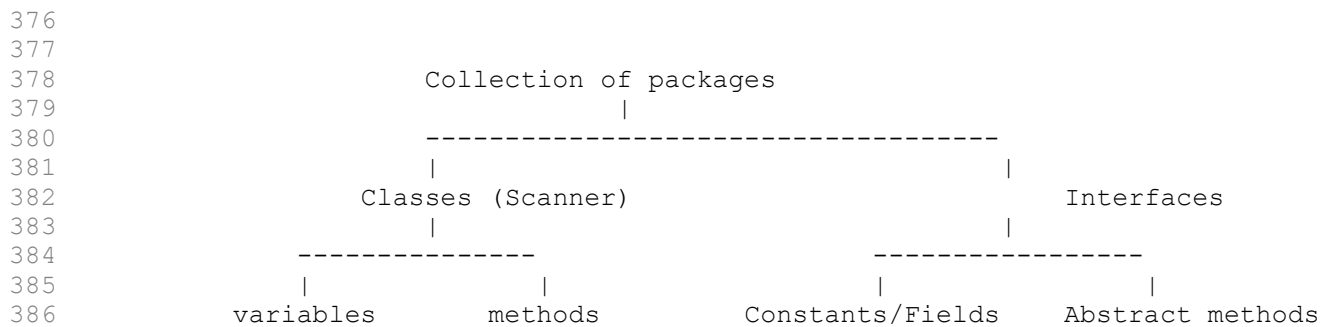
345
346 }
347
348 access-specifier or access label or visibility
349
350
351 public / private /protected / <default>
352
353 access-modifier ==> behaviour
354
355 static or final or <default>
356
357
358 class Sample
359 {
360
361 access-specifier access-modifier return-type methodName(arguments or parameters)
362 public static void main(String[] args)
363 {
364 //local declarations
365 //Execuatbale statements
366
367
368 }
369
370 }
371
372

```

```

373 -----
374 Java Naming Conventions
375 -----

```



```

389
390 packages : all lower case
391
392 package sep8th;
393
394 package java.util;
395
396 package regression;
397
398 package org.openqa.selenium;
399

```

```

400
401 -----

```

```

402
403 Class / Interface : Title Case
404

```

```

405
406 RamaKrishna RK
407
408 Hello
409
410 VariablesDemo
411
412 Scanner
413

```

```

414 ChromeDriver
415
416 -----
417
418 variables / methods / Abstract methods : camel case (LowerCase + Title case)
419
420
421 x y
422
423 height
424
425 netSalary
426
427 costOfItem
428
429 noOfLinks
430
431
432 Examples of methods :
433
434 methods (
435
436 nextInt(
437
438 getWindowHandle(
439
440 -----
441 Constants/Fields : All upper case
442
443
444 PI=3.14
445
446 MAX_LIMIT=10000;
447
448
449
450 *****
451 Variables in Java
452 *****
453
454 Variables
455
456
457
458
459 Local Instance Static
460
461
462
463 -> refers to a memory block
464
465 10 + 20
466
467 i=10 j=20
468
469 i+j
470
471 -> simplifies the expressions
472
473 -> stores ones recall anywhere in program
474
475
476 syntax :
477
478 access-specifier access-modifier datatype variableName ; Variable Declaration
479
480 access-specifier access-modifier datatype variableName = value ; Variable Declaration
 and assignment
481

```

```

482
483 int i=10
484
485 -----
486 Local Variable
487 -----
488
489
490 1. Defintion :
491
492 a variable which is declared inside a method is called Local
493
494 2. When ?
495
496 when memory is getting allocated ?
497
498 when method is called
499
500 3. where ?
501
502 where these variables are getting
503
504 Java Stack Area
505
506 4. calling style
507
508 just variable name
509
510 5. scope or life time
511
512 till the end of method or within that method
513
514
515 Note :
516
517 for Local Variables , it is the responsbility of developer/programmer to initialize some
value
518
519 VariablesDemo1.java:14: error: variable y might not have been initialized
520 System.out.println(y);
521 ^
522
523
524
525 1. Defintion :
526
527
528 2. When ?
529
530 when memory is getting allocated ?
531
532
533 3. where ?
534
535 where these variables are getting
536
537
538 4. calling style
539
540
541 5. scope or life time
542
543 -----
544 Instance Variable or non-static variable
545 -----
546 1. Defintion :
547
548 variables which are declared inside a class but outside of method
549

```



```
550 2. When ?
551
552 when memory is getting allocated ?
553
554 only when you create an instance / object of that class
555
556 how to create an instance or object ?
557
558 ClassName objectName=new ClassName();
559
560
561 3. where ?
562
563 where these variables are getting
564
565 in Heap Area
566
567
568 4. calling style
569
570
571 objectName.variableName;
572
573
574
575 5. scope or life time
576
577
578 till the last usage of object in the program
579
580 -----
581 Static variable
582 -----
583
584 1. Defintion :
585
586 variables which are declared inside a class but outside of method having keyword static
587
588 2. When ?
589
590 when memory is getting allocated ?
591
592 during class loading
593
594
595 3. where ?
596
597 where these variables are getting
598
599 Class Area /Method Area
600
601
602 4. calling style
603
604 ClassName.variableName;
605
606
607 5. scope or life time
608
609 till the end of program
610
611 -----
612 Data Types in Java :
613
614 Java is Strongly typed language
615
616
617 1996
618
```

```
619 8 4 2 1
620
621 3 0 0 0 0 0 0 1 1
622
623 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
624
625
626
627
```

## Data Types in Java

```
630
631 primitive (8) `
 non-primitive
```

```
632
633 boolean
634 char
635 byte
636 short
637 int
638 long
639 float
640 double
641
642
643
```

644 How to download eclipse ?

645 eclipse is an IDE (Integrated Development Environment)

646 IDE's like eclipse , Intelli J Idea , Vs code , net beans etc  
647 ..

648 <https://www.eclipse.org/downloads/>

649 Click on "Download Packages " link

650 under "MORE DOWNLOADS"

651 click on "Eclipse 2021-09 (4.21)"

652 under "Eclipse IDE for Enterprise Java and Web Developers"

653 click on "Windows x86\_64"

## Data Types in Java

```
665
666
667
668 Exception in thread "main" java.lang.Error: Unresolved compilation problem:
669 Type mismatch: cannot convert from int to byte
670 Type mismatch: cannot convert from int to short
```

```
671
672 at sep13th.DataTypesDemo1.main(DataTypesDemo1.java:9)
673
```

```
674 Exception in thread "main" java.lang.Error: Unresolved compilation problem:
675 The literal 2147483648 of type int is out of range
```

```
676
677 at sep13th.DataTypesDemo1.main(DataTypesDemo1.java:13)
678 -----
```

## Methods in Java

```
680 -----
681
682 Method :
```

683 a set of statements or instructions

685

```

686 need of methods :
687
688 - resusabilty
689 - reduces code redundancy
690 - increases code clarity
691 - code debugging becomes easy
692
693
694 syntax :
695
696
697 access-specifier access-modifier return-type methodName(parameters or arguments)
698 {
699 //local declarations
700 //statements
701
702 return statement ; if return-type is void no need of return satement
703
704 }
705
706
707
708
709

```

## Methods in Java

|            |        |
|------------|--------|
| Instance   | Static |
| or         |        |
| Non-static |        |

### Non-Static Methods

#### 1. Defintion :

method defined or declared without static keyword

<default> , abstract , transient , volatile etc ..

#### 2. When ?

when memory is getting allocated ?

creation of object

#### 3. where ?

where these variables are getting

Heap Area

#### 4. calling style

objectName.methodName();

#### 5. scope or life time

till last usage of object

????????????????????????????????

Write a Java Program to calcuate Simple Interest

using methods concept

????????????????????????????????

```
755
756 Static :
757
758
759 1. Defintion :
760
761 method defined or declared with static keyword
762
763
764 2. When ?
765
766 when memory is getting allocated ?
767
768 during .class file loading
769
770 3. where ?
771
772 where these variables are getting
773
774 Class Area /Method Area
775
776
777 4. calling style
778
779 ClassName.methodName();
780
781
782 5. scope or life time
783
784 till the end of program
785
786
787 =====
788 OOP Object Oriented Programming
789
790 POP Procedure oriented programing (c , COBOL etc)
791
792 Object
793
794 Class
795
796 Encapsulation
797
798 Polymorphism
799
800 Inheritance
801
802 Abstraction
803
804 Object :
805
806 anything that exists is an object
807
808 3 characteristics :
809
810 State : data or attributes that represents an object
811
812 variables
813
814 Behaviour : actions or tasks that object can perform
815
816 methods
817
818 Identity : unique id used by JVM to identify object
819
820
821 -----
822 Class
823
```

```

824 blueprint or template from which you can create an object
825
826 -----
827 Object Creation Technique
828 -----
829
830 1. using new operator
831
832 ClassName objectName=new ClassName();
833
834 2. using static factory method
835
836 ClassName objectName=ClassName.methodName();
837
838 3. using non-static factory method
839
840 will be used in case of dependency of a object of some class on another class
841
842 ClassName1
843 methodName : object of ClassName2
844
845 ClassName2
846
847 ClassName1 object1=new ClassName1();
848 ClassName2 object2=object1.methodName();
849
850 XSSFWorkbook
851
852 getSheet(): object of XSSFSheet
853
854 XSSFSheet
855
856 XSSFWorkbook wb=new XSSFWorkbook();
857 XSSFSheet ws=wb.getSheet();
858
859 4. using clone method
860
861 5. using String Literals
862
863 String s1="FLM";
864
865 6. using deserialization
866 (no syntax)
867 used by JVM to create object
868
869
870
871
872
873
874 Encapsulation :
875
876 Encapsulation in Java is a process of wrapping code and data together into a single unit,
877 for example, a capsule which is mixed of several medicines.
878
879
880 Encapsulation = Data hiding + Abstraction
881
882
883
884 The field Bank.balance is not visible
885
886 For Tightly Encapsulation , we can use access-specifier as private
887
888 if you make variables private only that class method can use it
889
890 then if you want to access the private variables of a class ,
891 create a public methods in that class and retrieve them using proper validation
892

```

```

893 //getters
894
895 methods having prefix get
896
897 these are used to retrieve data or variables
898
899
900 //setters
901
902 methods having prefix set
903
904 these are used to modify or alter data or variables
905 -----
906 this keyword
907 -----
908
909 if we have local variables same as instance variables then local variables dominates
910 to overcome this problem , we have keyword "this" in java
911
912 this :
913
914 is keyword cum operator which is used for current calling object reference
915
916 *****
917 Java Season - 1
918 Day 10 - 16th September - Saturday
919 *****
920
921 -----
922 Polymorphism
923 -----
924
925 Poly means many
926
927 morph means forms
928
929 Exception in thread "main" java.lang.Error: Unresolved compilation problem:
930 The method add(int, int) in the type Sum is not applicable for the arguments
931 (double, double)
932 at sep16th.SumDemo.main(SumDemo.java:12)
933
934 The method add(int, int) in the type Sum is not applicable for the arguments (long, long)
935
936 -----
937 without Polymorphism
938
939 public void add(int x,int y)
940 {
941 System.out.println("Sum is "+(x+y));
942 }
943
944 public void addDouble(double x,double y)
945 {
946 System.out.println("Sum is "+(x+y));
947 }
948
949 public void addLong(long x,long y)
950 {
951 System.out.println("Sum is "+(x+y));
952 }
953
954 -----
955
956 with polymorphism
957
958 public void add(int x,int y)
959 {
960 System.out.println("Sum is "+(x+y));

```

```

961 }
962
963 public void add(double x,double y)
964 {
965 System.out.println("Sum is "+(x+y));
966 }
967
968 public void add(long x,long y)
969 {
970 System.out.println("Sum is "+(x+y));
971 }
972

```

```

973 -----

```

## 978 Polymorphism

|     |               |              |
|-----|---------------|--------------|
| 982 | Static        | Dynamic      |
| 983 | or            | or           |
| 984 | Compile time  | Runtime      |
| 985 | or            | or           |
| 986 | Early Binding | Late Binding |

989 binding means mapping or linking

991 the caller and called method ,

993 In Static polymorphism or method overloading

995 binding will happen during compile time ,

997 Static Polymorphism :

999 can be achieved by means by

1000 Operator Overloading (not encourage or not supported in java)

1003 Method Overloading

1005 Dynamic Polymorphism :

1007 can be achieved by means by

1009 method overriding (this will be discussed after inheritance topic)

```

1010 -----

```

1012 Method Overloading

```

1013 -----

```

1016 method signature

1018 methodname + Paramaters / arguments

1020 in Method Overloading in same class

1022 we will have same method name and with different signatures

1025 Method Overloading -- different data types

1027 -- change in no of arguments

1028

```

1029 -- change in order of arguments (not preferred)
1030
1031
1032 *****
1033 PrintStream Class ==> java.io
1034
1035 4 print methods
1036
1037 println() : this will print and takes the cursor to new line
1038
1039 print(): this will print and stays cursor in same line
1040
1041 *
1042 * *
1043 * * *
1044 * * * *
1045 * * * * *
1046
1047 5 5 5 5 5
1048 4 4 4 4
1049 3 3 3
1050 2 2
1051 1
1052
1053 printf() : from previous lang
1054 printf(String, Object...)
1055
1056
1057
1058 format()
1059
1060
1061 _____
1062
1063 git repo for programs
1064
1065
1066 git repo for programs
1067
1068 https://github.com/flm31stjuly/JavaSeasons
1069
1070 ???
1071
1072 use method overloading concept for below requirment
1073
1074
1075 2,10,1869 - > 02-10-1869
1076
1077 1,1,500 -> 01-01-0500
1078
1079 ???
1080
1081 -----
1082 Constructor :
1083
1084 def : it's method having same name as class name
1085
1086 rules of constructor :
1087
1088 1) Constructor name should be same class name (case sensitive)
1089 2) Constructor shouldn't have return-type
1090 if return-type is mentioned .then what ?
1091 then compiler will treat as normal method
1092 3) Constructor shouldn't have access-modifier
1093 if access-modifier is specified , then compilation error
1094
1095 Illegal modifier for the constructor in type Sample; only public, protected & private
are permitted
1096

```



```

1097
1098 Constructor
1099
1100
1101 default constructor parameterized
 constructor
1102
1103 Constructor Overloading :
1104
1105
1106
1107
1108 -----
1109 3.Inheritance
1110 -----
1111
1112
1113 class B extends A
1114
1115 1. Single or Simple
1116 2. MultiLevel
1117 3. Hiererichal
1118 4. Multiple
1119 5. Hybrid
1120
1121 Marks extends Student
1122
1123 child Parent
1124 derived base
1125
1126
1127 Java doesn't allow Multiple Inheritance for classes
1128
1129 public class C extends A,B{
1130
1131 }
1132
1133
1134 -----
1135 Method Overiding
1136 -----
1137
1138 method signature
1139
1140 methodname + Paramaters / arguments
1141
1142
1143 methods having same signature in two different classes having relationship
1144
1145 *****
1146 Java Season - 1
1147 Day 14 - 23rd September - Saturday
1148 *****
1149
1150 -----
1151 4. Abstraction
1152 -----
1153
1154 Encapsulation = Data Hiding + Abstraction
1155
1156 Abstraction
1157
1158
1159
1160 Abstract class Interface
1161
1162 (0 to 100 %) 100 %
1163
1164

```

```

1165
1166 abstract methods :
1167
1168 syntax :
1169
1170 access-specifier abstract return-type methodName(arguments); ==> idea and no
implementation
1171
1172
1173 Note :
1174
1175 for abstract class we Cannot instantiate/create objects
1176
1177
1178 Interface : (100 % abstraction)
1179
1180 constant / fields and abstract methods
1181
1182
1183
1184 Methods :
1185
1186 1) what is the arguments that method is taking
1187 2) what is the return-type
1188 3) what does this method do ?
1189
1190
1191
1192
1193 operators:
1194
1195 special symbols and it peforms particular operation
1196
1197 based on no of operands ,
1198
1199 operators
1200
1201
1202
1203 unary binary ternary
1204
1205
1206 operand can be a value or variable
1207
1208 int x=2;
1209 int y=3;
1210
1211 z = x+y
1212
1213
1214
1215 2 + 3
1216
1217 2,3 operands
1218 + operator
1219
1220 unary :
1221
1222
1223
1224 ++ --
1225
1226
1227 post pre
1228
1229 inc dec inc dec
1230
1231 a++ a-- ++a --a
1232

```

```

1233
1234
1235
1236
1237 int x=20;
1238
1239 x++;
1240
1241 x--;
1242
1243 ++x;
1244
1245 --x;
1246
1247
1248
1249 Binary operators :
1250
1251
1252
1253 - Aritmetic operator / Mathematical
1254
1255 - Realtional Operators
1256
1257 - Logical Operators
1258
1259 - Bitwise Operators
1260
1261 - assignment operators
1262
1263
1264 - Aritmetic operator / Mathematical :
1265
1266 + - * / %
1267
1268
1269 - Realtional Operators
1270
1271 comparision between two things
1272
1273
1274
1275 > < >= <= != == ==> true or false
1276
1277
1278 Logical Operators:
1279
1280 b/n two comparisions
1281
1282
1283 && || !
1284
1285 AND OR NOT
1286
1287
1288
1289 Bitwise Operators:
1290
1291 & | ~ ^ >> << >>>
1292
1293 AND OR tilt XOR Right left unary
1294 shift shift right shifter
1295
1296
1297 Assignment :
1298
1299 = += -= *= /= %= >>= <<=
1300
1301 x=10;

```

```

1302
1303
1304 if x= 10
1305 x+=10; => x=x+10;
1306
1307
1308
1309 x=12; 1100
1310
1311 y=4; 0100
1312
1313 -----
1314 1100
1315
1316
1317
1318
1319
1320
1321 Scanner Class :
1322
1323
1324 Scanner(InputStream)
1325
1326 here InputStream object is nothing but System.in
1327
1328
1329 nextInt():int
1330
1331 nextLong() : long
1332
1333 nextShort() : short
1334
1335 nextByte() : short
1336
1337 nextDouble() : double
1338
1339 nextFloat(): float
1340
1341 nextBoolean() : boolean
1342
1343 next() : String
1344
1345
1346
1347 Ternary operators : (conditional operators or decision making operations)
1348
1349 ?:
1350
1351 syntax :
1352
1353 expr1 ? expr2 : expr 3
1354
1355 expr1 condition or comparision
1356
1357 expr2 True part
1358
1359 expr3 False part
1360
1361
1362
1363 Flow Control or Control flow in Java :
1364
1365
1366 if else
1367
1368 if(boolean value or an expression which results in boolean or condition)
1369 statement1; // will be excuted when condtion is true
1370
1371 else

```

```

1371 statement2; // will be excuted when condtion is false
1372
1373 if(boolean value or an expression which results in boolean or condition)
1374 {
1375
1376 //block of statements
1377 }
1378 else
1379 {
1380 //block of statements
1381 }
1382
1383
1384 https://www.facebook.com/
1385
1386
1387 // <= 30000 20%
1388 // >30000 <=60000 30%
1389 // >60000 40%
1390
1391
1392 Non-IT IT
1393
1394
1395 Switch:
1396
1397
1398 syntax
1399
1400
1401 switch(variable/expression)
1402 {
1403
1404 case value1 : statements
1405 break;
1406 case value2 : statements
1407 break;
1408
1409
1410 default : statements;
1411
1412
1413 }
1414
1415
1416 byte short int char String (from 1.7)
1417
1418
1419 Not allowed : long float double boolean
1420
1421
1422 Loops or Iterative statements
1423
1424
1425
1426 i) for looping
1427
1428 syntax :
1429
1430
1431 for(counter intialization; condition check ; increment or decrement)
1432 {
1433
1434 //statements or code
1435
1436
1437 }
1438
1439

```

```

1440 ii) while loop
1441
1442 while(condition check)
1443 {
1444
1445 //statements or code
1446
1447 }
1448
1449
1450 iii) do while
1451
1452
1453 iv)for each or enhanced for loop
1454
1455
1456 syntax :
1457
1458 for(datatype someReferenceVaribale: arrayName or Collection refernce)
1459 {
1460
1461 //use the someReferenceVaribale means the value
1462 }
1463
1464 -----
1465 Arrays :
1466
1467
1468 int x=10;
1469
1470 int y=20;
1471
1472
1473 def : indexed collection of homogenous datatype elements having fixed length
1474
1475
1476 syntax :
1477
1478 method 1:
1479 datatype[] arrayName={value1,value2,value3..valuen}
1480
1481 datatype[] arrayName ==> declaration
1482
1483 arrayName={value1,value2,value3..valuen} ==> initialization
1484
1485 length : int (this is not method)
1486
1487
1488 method 2:
1489
1490 using new
1491
1492
1493 datatype[] arrayName=new datatype[length of array or no of elements in array];
1494
1495
1496 *****
1497 Java Season - 2
1498 Day - 5th October - Thursday
1499 *****
1500
1501
1502
1503 datatype[] arrayName=new datatype[length of array or no of elements in array];
1504
1505 dis advantages :
1506
1507 1. arrays always allow similar datatype
1508 2. arrays doesn't have growable nature

```

```

1509
1510 Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of
1511 bounds for length 5
1512 at oct5th.ArraysDemo5.main(ArraysDemo5.java:15)
1513 3. data structured related predefined methods are not available
1514
1515 Collections :
1516
1517
1518
1519
1520 Collection(I)
1521
1522
1523 List(I) Queue(I) Set(I)
1524
1525 - ArrayList(C) - HashSet()
1526 - LinkedList(C) - LinkedHashSet(C)
1527 - Vector(C) - SortedSet(I)
1528 - Stack(C) - TreeSet(C)
1529
1530
1531 Collection(I)
1532
1533 add(E):boolean
1534 addAll(Collection<? extends E>):boolean
1535 contains(Object):boolean
1536 isEmpty():boolean
1537 size():int
1538
1539
1540 List(I):
1541
1542 add(int, E):void
1543 get(int):E
1544 indexOf(Object):int
1545 toArray():Object[]
1546
1547
1548 ArrayList(C) :
1549
1550 ArrayList()
1551 ArrayList(int)
1552 ArrayList(Collection<? extends E>)
1553
1554 subList(int, int):List<E>
1555
1556
1557 *****
1558 Java Season - 2
1559 Day - 56h October - Friday
1560 *****
1561
1562
1563 Iterator :
1564
1565 iterator():Iterator<E>
1566
1567 2 methods :
1568
1569 hasNext():boolean
1570
1571 next():E
1572
1573 this will do 2 things
1574
1575 - retrieve the element
1576 - moves the cursor to next level

```

```

1577
1578 -----
1579 Set(I):
1580
1581 No get method here , because insertion order(index) is not preserved
1582
1583
1584
1585 Constructors:
1586
1587 HashSet()
1588 HashSet(int)
1589 HashSet(int, float)
1590 HashSet(int, float, boolean)
1591 HashSet(Collection<? extends E>)
1592
1593 Here float is meant for Load factor
1594
1595
1596
1597

```

| List<br>(ArrayList)                                   | vs | Set<br>HashSet                                                     |
|-------------------------------------------------------|----|--------------------------------------------------------------------|
| - insertion order is preserved<br>means index         |    | - insertion order is not preserved<br>because of hashing technique |
| - duplicates are allowed                              |    | - duplicates are not allowed                                       |
| internal structure or implementation<br>- linked list |    | - HashMap                                                          |
| to retrieve the elements                              |    |                                                                    |
| - get(index)                                          |    | we should use for-each or iterator                                 |

```

1615 -----
1616 Exception Handling :
1617
1618
1619 Runtime Error is called Exception .
1620
1621
1622
1623 try catch finally throw throws
1624
1625
1626 Exception in thread "main" java.lang.ArithmeticException: / by zero
1627 at oct7th.ExceptionDemol.main(ExceptionDemol.java:15)
1628
1629
1630 10
1631
1632
1633 try
1634 {
1635
1636 // risky code
1637
1638 }
1639 catch(argument) //here argument is an exceptionclass with reference
1640 {
1641
1642 //handling code or info code
1643 }
1644 finally
1645 {

```



```

1646 // some code to be excuted irrespective of exception
1647 //compulsory
1648 }
1649
1650
1651 try finally
1652
1653 try catch
1654
1655 try catch finally
1656
1657 try catch catch
1658
1659 try catch catchfinally
1660
1661
1662

```

## Exception Types

1)Checked Exceptions

2)Unchecked Exceptions

MEthods in Throwable or Exception Class :

```

1674 getMessage():String
1675
1676 printStackTrace():void
1677
1678

```

Note : while using multiple catch blocks you should follow bottom to top hiererchy

Unreachable catch block for ArithmeticException. It is already handled by the catch block for RuntimeException

```

1682 -----
1683

```

throw

used to create and throw user-defined or custom exception

<https://github.com/flm31stjuly>

break statement:

used in loops , switch

Exception in thread "main" java.lang.Error: Unresolved compilation problem:  
Unreachable code

at oct9th.BreakDemo2.main(BreakDemo2.java:16)

continue statement :

use in loop