JAVA Important Points From Revision(Part - 1)

1)- With the help of JDBC we can store data in database. With IO operations we can store it in HDD and in microservices we can store it in cloud.

2)- Java uses Hybrid mode of execution

3)- Char in java is of 2 bytes due to Internationalization(in order to cover the characters of different languages.)

4)- WORA --> Write once run everywhere(Java --> Platform independent while JDK is Platform dependent). We can upload an application in the internet and users can use that application on their devices by using JDK(Platform dependent).

5)-In java to create an object for a class we have "new" keyword, but we don't have delete keyword to destroy the object where as destroying the object is taken care by a program(thread[daemon thread]) called "GarbageCollector".

6)- Every assignment should be checked by the compiler for the type compatibility, so we say java is "Strictly typed programming language".**(+ dependent on implicit and explicit type casting - left for now. also + in arithmetic operators → Note: if we apply arithmetic operators b/w 2 variables then the result type is always max(int,typeof a, typeof b))**

> **Note:** In java language, the memory for the variable will be given at the runtime , so we say java language as "Dynamic programming language".

7)- How many keywords are there in java?

Ans. 53

Diagrams to cover ->

1)- Memory mapped diagram

2)- Hierarchy of object

8)- javac Test.java → Inform compiler to compile the file Test.java.

java Test → Inform JVM to load the Test.class(which have main method) file and start the execution.

9)- Different cases in conditional statements

a)-

```
int x=0;
if(x){
    System.out.println(x);
}
```

output:Compile time error(x should be a boolean(error: incompatible types: int cannot be converted to boolean))

b)- if(true) int x =10; **//CE**

c)- if(true){ int x =10;} System.out.println(x); **//CE**

d)- if(true){ int x =10;}. **//No CE**

e)-

```
switch(x){
    ...
}
```

x could be —> byte , short , char, int, Byte , Short , Character , Integer , enum , String

f)-

```
int x= 10;
switch (x){
    System.out.println("hiee");
}
```

Output: CE(If we are writing any statement inside switch then those statements should be a part of case or default.)

g)-

```
int y= 20;
switch (x){
    case 10: System.out.println(10);
    case y: System.out.println(20); //CE: constant required
}
```

h)-

final -> This keyword makes the variable compile time constant, so compiler will be knowing the value of the variable.

```
int x= 10;
final int y= 20;
switch (x){     //byte,short,int,char
    case 10: System.out.println(10);
    case y: System.out.println(20);
    case 10+20+30:  System.out.println(60); //can have expressions as well(but should be constant)
}
```

Output: Code will be compiled

i)- **Note:** In java language, the memory for the variable will be given at the runtime , so we say java language as "Dynamic programming language".

j)- case label values should lie in the range of switch argument type, otherwise it results in "Compiletime Error".

```
byte b= 10;
switch (b){ //byte = -128 + 127
    case 10 : System.out.println(10);
    case 100 : System.out.println(100);
    case 1000 : System.out.println(1000);//CE: not with in the range of byte
```

}

k)-

```
byte b= 10;
switch (b+1) { //byte + int => int
    case 10 : System.out.println(10);
    case 100 : System.out.println(100);
    case 1000 : System.out.println(1000);
}
```

Output: No Output(as b+1 is an integer not byte)

l)-

```
int x = 10;
switch (x){ //byte---int, short---int, char('a')---int(97), int
    case 97:System.out.println("97");
    case 99:System.out.println("99");
    case 'a':System.out.println("100");
}
```

Ouput: CE: duplicate case labels are not allowed.(`a` —> 97)

m)- Conclusion

a. Case label should be compile time constant.

b. Case label can have expression, but it should be a compile time constant expression.

c. Case label value should be within the range of the switch argument type.

d. Duplicate case label are not allowed.

n)-

```
int x = 3;
switch (x){
    default: System.out.println("default");
    case 0: System.out.println("0");
    case 1: System.out.println("1");
    break;
    case 2: System.out.println("2");
}
```

Output

x=3

default

0

1

(VERY VERY IMP —> As no break was present both 0 and 1 are also printed —> Fall Through in Switch (weird as case values are different))

o)-

```
int x = 3;
switch (x){
    default: System.out.println("default");
    default: System.out.println("default");
}
```

Output: CE: duplicate default label.

p)-Note: The argument to a while statement should be of "boolean type".if we are using anyother type it results in "CE".

i)-

```
while (1)
{
    System.out.printl("hello");   //CE
}
```

ii)-

```
while (true)
    int x= 10; //CE: delcaration not allowed
```

iii)-

```
while (true)
;
```

Output: no output(valid)

iv)-

```
while (true)
{
    int x =10; //valid
}
```

v)-

```
while (true)
{
    int x =10;
}
System.out.println(x);    //CE: can't find symbol.
```

q)- Note::

=> Every final variable will be replaced with corresponding value by the compiler

=> if any operation involves only constants then compiler is responsible to perform operation.

=> if any operation involves at least one variable, then compiler won't perform any operation, jvm will perform that operation.

eg#1.

```
while (true)
{
```

```
    System.out.println("hello");
}
System.out.println("hiee");  //CE: unreachable code
```

eg#2

```
while (false)
{
    System.out.println("hello");   //CE: unreachable code
}
System.out.println("hiee");
```

eg#3.

```
int a= 10;
int b= 20;
while (a<b).  //JVM :: 10<20 -> true
{
    System.out.println("hello");
}
System.out.println("hiee");
```

Output: hello(infinite times)

eg#4.

```
final int a= 10;
final int b= 20;
while (a<b).  //JVM :: 10<20 -> true
{
    System.out.println("hello");
}
System.out.println("hiee");  //unreachable code
```

Output: CE

eg#5.

```
final int a= 10;
while (a<20)
{
    System.out.println("hello");
}
System.out.println("hiee");  //unreachable code
```

Output: CE

eg#6.

```
int a= 10;
while (a<20)
{
```

```
    System.out.println("hello");
}
System.out.println("hiee");
```

Output: hello(infinite times)

All the above unreachable code scenarios are also for do-while and for

r)- do-while

eg::

```
do{
    System.out.println("hello");
}while(true);
```

output:: hello infinite times

eg::

```
do;while(true);
```

output:: compiles succesfull

eg::

```
do
    int x=10;
while(true);
```

output:: compile time error

eg::

```
do{
    int x=10;
}while(true);
```

output:: compilation succesfull

eg::

```
dowhile(true);
```

output:: compilation error.

s)- for loop —>

eg::

```
int i=0;
for(System.out.println("hello");i<3;System.out.println("hi")){
    i++; // i = 1,2,3
}
```

Output:: hello hi hi hi

eg:: //by default condition is kept as true by compiler(also consider unreachable for the same case)

```
for(;;){
    System.out.println("hello");
}
```

Output:: hello(infinite times)

eg::

```
for(;;)
    int x=10;
```

output:: CE

eg::

```
for(;;){
    int x=10;
}
```

output:: no output

eg::

```
for(int i=0;i<true;i++){
    System.out.println("sachin");
}
System.out.println("dhoni");
```

output::CE (I is integer and second operand after < is of type boolean)

s)-

```
for (int i = 0; i <= 10; i++) {
    if (i > 6) break;
}
System.out.println(i);
```

Compilation fails.[Answer: i not accesible outside for loop]

t)-

```
public class Test {
    public static void main(String[] args) {
        boolean b1 = 0;
        boolean b2 = 1;
        System.out.println(b1 + b2);
    }
}
```

compilation error[Answer: boolean means only true,false]

u)-

```
public static void main(String[] args) {
    boolean i = true; //line-12
    switch(i) { //line-13
        case true: System.out.println("true"); break;
        case false: System.out.println("false"); break;
        default: System.out.println("other"); break;  //line-15
    }
}
```

Compilation fails because of an error on line 13.[Answer: switch arg types can be : byte,short,int,char]

10)- Break Statement ->

a. Inside switch to avoid fallthrough

b. Inside loops to break the loop based on some condition

c. Inside label block to break label block execution based on some condition.

d. break can be used along with blocks also

```
int x= 10;
l1:{
    System.out.println("begin");
    if (x==10)
        break l1;
    System.out.println("end");
}
System.out.println("hello");
```

Output

begin

hello

e)- Note: break can be used inside "switch or loop", any other places if we try to use it results in "CompileTime-Error." (same for continue —> can only be used in loops)

```
int x= 10;
System.out.println("hello");
if (x==10)
break;
System.out.println("hiee");
```

11)-Note: Compiler won't check for unreachablity in case of "if-else" statement, whereas it checks for unreachability in case of loops.

eg:

```
if (true)
    System.out.println("hello");//hello
else
    System.out.println("hiee");
```

12)-Operators ->

a)- Increment and Decrement →

i)- int y = 10++; **//CE(increment or decrement can be applied only on variables, but not on values directly.)**

ii)- we can't perform nesting of increment or decrement operator, it would result in compile time error.

```
int x= 10;
int y= ++(++x); //CE
```

iii)- For a final variable, increment or decrement operation can't be done.

```
final int x= 4;
x++; //CE
```

iv)- we can't apply increment or decrement operator on boolean type, where as it can be applied on other primitive types.

```
int x= 10;
x++;
System.out.println(x);//11
char ch='a';
ch++;
System.out.println(ch);//b
double d = 10.5;
d++;
System.out.println(d);//11.5
boolean b= true;
b++; //CE
System.out.println(b);
```

v)- What is the difference b/w b = b+1 and b++?

```
byte b= 10;
b = b+1; //CE (byte+int :: int)
System.out.println(b);
byte b = 10;
b++; // b = (byte)(b+1);
System.out.println(b);//11
```

b)- Arithmetic operator —>

i)- Note: if we apply arithmetic operators b/w 2 variables then the result type is always max(int,typeof a, typeof b)

eg:

byte + byte = int

byte + short = int

int + double = double

char + char = int

char + double = double

eg#1.

```
System.out.println('a' + 'b');  //195
System.out.println('a' + 1);  // 98
System.out.println('a' + 1.2);  //98.2
```

ii)- In integral arithmetic(byte , short , int , long) there is no way to represent "Infinity", so result will be "ArithmeticException".

In case of double, float types, there is a possibility of storing "Infinity" so the result will be "Infinity".

```
System.out.println(10/0.0); // int/double = +double
System.out.println(-10/0.0); // -int/double = -double
System.out.println(0/0); // int/int = int
```

output

Infinity

-Infinity

ArithmeticException :/by zero

iii)- NAN

a)- Nan(Not a Number) is a integral arithmetic(byte , short , int , long) there is no way to represent the undefined result, so it would throw an Exception called "ArithmeticException".

b)- But floating point arithmetic(double , float) there is a way to represent the undefined result, so the result would be "Nan".

```
System.out.println(0.0/0.0); //double/double -> double
System.out.println(-0.0/0.0);//double/double -> double
System.out.println(0/0); //int/int -> int
```

Output

Nan

Nan

ArithmeticException : /by zero

Important Points —>

1)- 0.0/0.0 —>NAN, while 1.0/0.0 —> infinity

2)- Note: for any value of 'x' including NaN, the result will be false.

```
System.out.println(10<Float.NaN);//false
```

```
System.out.println(10<=Float.NaN);//false
System.out.println(10>Float.NaN);//false
System.out.println(10>=Float.NaN);//false
System.out.println(10==Float.NaN);//false
System.out.println(Float.NaN == Float.NaN);//false
System.out.println(10!=Float.NaN);//true
System.out.println(Float.NaN != Float.NaN);//true
```

3)- '+' operator in java is reffered as "Overloaded-Operator". '+' operator will perform addition if both the operands are of numeric type(byte , short , int , long , float , double) . '+' operator will perform concatenation, if one of the operand is of "String" type.

c)-RelationalOperator in Java

<,<=,>,>=

=> We can apply relational operators only on primtive types except "boolean types".

=> we cannot apply relational operators on reference types(on objects)

=> Nesting of relational operator is not possible in java.

eg#1.

```
System.out.println(10>10.5);//false
System.out.println('a'>10.5);//true
System.out.println('z'>'a');//true
System.out.println(true>false);//CE
System.out.println("sachin">"kohli");//CE
System.out.println(10<20<30);//CE
System.out.println(10>20>30);//CE
```

d)- EqualityOperator

==,!=

=> We can apply equality operators on primitive types including boolean types

eg#1.

```
System.out.println(10 == 20);//false
System.out.println('a' == 'b');//false
System.out.println('a' == 97.0);//true
System.out.println(false == false);//true
```

=> we can apply equality operators on reference type also.

eg#2.

```
Thread t1= new Thread();
Thread t2= new Thread();
Thread t3= t1;
System.out.println(t1==t2);//false
System.out.println(t1==t3);//true
```

Note:

=> To use == operator on reference type, we need to check whether there exists a relationship b/w 2 operands.

=> If relationship exists, it should be parent-child relationship, otherwise it would result in "CompileTimeError".

eg#3.

```
Thread t = new Thread();
Object o = new Object();
String s = new String("sachin");
StringBuffer sb =new StringBuffer("dhoni");
System.out.println(t==o);//false
System.out.println(o==s);//false
System.out.println(s==t);  //CE
System.out.println(s==sb);  //CE
```

e)- Bitwise Operator

1)- &,|,^ => These operators can be applied on boolean and even on integral types.

2)- ~(bitwsise complement) => This operator can be applied on integral types, but not on boolean types.

3)- !(boolean complement) => This operator can be applied only on boolean types, but not on integral types.

Difference between &,| and &&,||(ShortCircuit Operator)

a)- &,|

=> Both the arguments should be evaluated always

=> Performance is relatively slow.

=> It is applicable for both integral and boolean types.

b)- &&,||(ShortCircuit Operator)

=> Second argument evaluation is optional.

=> Performance is relatively high

=> It is applicable only for boolean types, not for integral types.

Type casting operator

1. implicit/narrowing => compiler will automatically do(no loss of data)
2. explicit/widening => programmer should do(loss of data might happen)

=> if we work with floating point data and if we try to assign it to integral type using explicit typecasting then the data after decimal value will be lost.

eg#1.

```
float x = 150.1234f;
int i =(int) x;
```

System.out.println(i)**;//150**

double d = 130.456;

int j = (int)d;

System.out.println(j);**//130**

=> While working with integral types, storing higher value in lower type using explicit typecasting might lead to data loss.

eg#1.

int x = 150;

short s =(short) x;

System.out.println(s);//150

//minRange + (result-maxRange-1)

/*

=-128 + (150-127-1)

=-128 + 150-128

= -128+22

= -106

*/

byte b = (byte)x;

System.out.println(b);//-106

eg#2.

double d= 130.456;

int x= (int)d;

System.out.println(x);//130

//minRange + (result-maxRange-1)

/*

=-128 + (130-127-1)

=-128 + 130-128

= -128+2

= -126

*/

byte b = (byte)d;

System.out.println(b);//-126

e)- Assignment Operator

There are 3 types of assignment operator

a. Simple assignment

eg: int a= 10;

int b= 20;

b. Chained assignment

c. Compound assignment

Chained assignment

eg#1

int a,b,c,d;

a= b= c= d= 10; **//valid**

eg#2.

int a=b=c=d=10; **//invalid**

System.out.println(a+" " + b + " "+c+" "+d);

eg#3.

int b,c,d;

int a=b=c=d=10; **//valid**

System.out.println(a+" " + b+" " + c +" " +d);

eg#4.

int a,b,c,d=10; // only d value is initialized

System.out.println(d);//10

f)- Compound assignment

+= ,-=, /=, *=, %=

&=,|=,^=

Note: In case of compound assignment operator, internally type casting will be performed automatically by the JVM similar to increment or decrement operator.

eg#1.

int a= 10;

a+=20;

System.out.println(a);

eg#2.

byte b=10;

b = b+1; **//incompatible types: required : int,found : byte**

System.out.println(b);

eg#3.

byte b = 10;

b++; **// b= (byte)(b+1);**

System.out.println(b);//11

eg#4

byte b=10;

b+=1; **//b = (byte)(b+1);**

System.out.println(b);//11

eg#5.

int a,b,c,d;

a=b=c=d=20;

a+=b-=c*=d/=2;

/*

d= d/2; 20/2 = 10

c= c*d; 20*10 = 200*

b= b-c; 20-200 =-180

a= a+b; 20-180 =-160

*/

System.out.println(a+" " +b +" " +c +" " +d);

g)- Conditional Operator

It is also called as "Ternary operator".

syntax: condition ? true : false

eg#1.

int x= 10==10 ? 100 : 500;

System.out.println(x); **//100**

eg#2.

final int a= 10,b= 20;

byte c1= (a > b) ? 30: 40**; // byte c = 40;**

eg#3.

int a1= 10,b1= 20;

byte c2= (a1 > b1) ? 30: 40; **//CE**

byte d2= (a1 < b1) ? 30: 40; **//CE**

System.out.println(c2 + " " + d2);

SNIPPET —>

1)- What will be the result of compiling and executing Test class?

```
public class Test {
    public static void main(String[] args) {
        System.out.println(1 + 2 + 3 + 4 + "Hello"); // 10Hello
        System.out.println( "Hello" + 1 + 2 + 3 + 4); //Hello1234
    }
}
```

2)-

```
public class Test {
    public static void main(String[] args) {
        System.out.println("Output is: " + 10 != 5);
    }
}
```

Ans —> compilation error

In Java, the + operator has a higher precedence than the != operator. Therefore, the expression is effectively treated as:

("Output is: " + 10) != 5

This leads to a compilation error because you are trying to compare a String with an int, which is not allowed.

To fix this issue, you need to use parentheses to explicitly define the order of operations:

"Output is: " + (10 != 5)

How many keywords are there in java?

Ans. 53

Reserve words meant for datatypes

a. byte,short,int,long

b. float,double

c. char

d. boolean

Reserve words for access modifiers(11)

a. public

b. private

c. protected

d. static

e. strictfp

f. synchornized

g. abstract

h. native

i. transient

j. volatile

k. final

Execution of Java Program(behind the scenes)

```
class Student
{
    //properties/fields/attributes
    String name;
    int age;
    //methods/behaviours
    public void dispStdDetails()
    {
        System.out.println(name);
        System.out.println(age);
    }
}
class Test
{
    public static void main(String[] args)
    {
        Student s = new Student();
        s.dispStdDetails();
    }
}
```

D:\OctBatchMicroservices>javac Test.java

Upon compilation of Test.java file 2 .class files will be loaded

a. Test.class[main()]

b. Student.class

D:\OctBatchMicroservices>java Test

JVM will load Test.class file by searching in the CWD[D:\OctBatchMicroservices>]

JVM found Test.class file which contains main() so JVM started the execution

JVM will split JRE into 3 regions

a. MethodArea[To keep .class file details]

b. StackArea[To keep the method body for execution]

c. HeapArea[To keep object data with default values for properties based on datatype]

Upon loading the main(), JVM started the execution

a. loaded main() body to StackArea.

b. Student s = new Student();

|=> JVM will go to heap area, create an object.

|=> Load Student.class file by searching in CWD[D:\

OctBatchMicroservices>] to MethodArea

|=> Depending upon the datatypes of properties, supply the

default values by giving a memory inside heaparea.

|=> Address of the object will be returned to the user.

c. s.dispStdDetails()

|=> JVM will call the method

|=> Body of dispStdDetails() will be loaded into stack area

|=> Execution of statements present inside dispStdDetails will

happen

S.o.p(name); S.o.p(age);

|=> Print the details of name and age into console[monitor]

d. Remove stackarea of dispStdDetails()

e. Remove stackarea of main()

d. Since Student Object doesn't have refernce, it becomes garbage object[Call

GarbageCollector[internally done by JVM]]

f. Unload the .class files from MethodArea[Test.class,Student.class]

g. Remove JRE /JVM from RAM.

Note: JVM by default will always search for main() with the following signature

public static void main(String[] args)

D:\OctBatchMicroservices>javac Test.java

Upon compilation of Test.java file 2 .class files will be loaded

a. Test.class[main()]

b. Student.class

D:\OctBatchMicroservices>java Test

JVM will load Test.class file by searching in the CWD[D:\OctBatchMicroservices>]

JVM found Test.class file which contains main() so JVM started the execution

JVM will split JRE into 3 regions

a. MethodArea[To keep .class file details]

b. StackArea[To keep the method body for execution]

c. HeapArea[To keep object data with default values for properties based on datatype]

Upon loading the main(), JVM started the execution

a. loaded main() body to StackArea.

b. Student s = new Student();

|=> JVM will go to heap area, create an object.

|=> Load Student.class file by searching in CWD[D:\ OctBatchMicroservices>] to MethodArea

|=> Depending upon the datatypes of properties, supply the default values by giving a memory inside heaparea.

|=> Address of the object will be returned to the user.

c. s.dispStdDetails()

|=> JVM will call the method

|=> Body of dispStdDetails() will be loaded into stack area

|=> Execution of statements present inside dispStdDetails will happen S.o.p(name); S.o.p(age);

|=> Print the details of name and age into console[monitor]

d. Remove stackarea of dispStdDetails()

e. Remove stackarea of main()

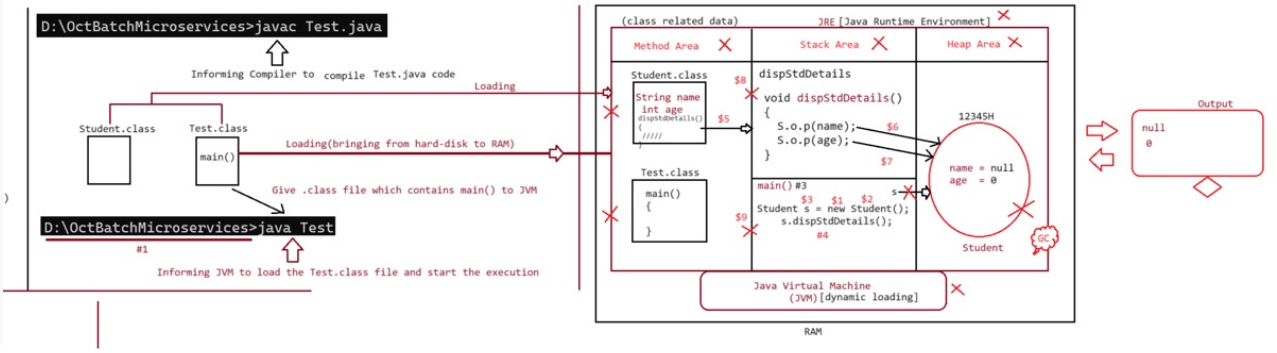
d. Since Student Object doesn't have refernce, it becomes garbage object[Call GarbageCollector[internally done by JVM]]

f. Unload the .class files from MethodArea[Test.class,Student.class]

g. Remove JRE /JVM from RAM.

Note: JVM by default will always search for main() with the following signature

public static void main(String[] args)



Objects —>

1)- new is an operator which is used to create an object/instance for a class in java.

2)- we don't have delete operator in java to destroy the objects, where destroying the objects is taken care by **"JVM(GarbageCollector)".**

++

22 —> snippets are not done as it includes & , |

++

Why we need Arrays?

To store mulitple values of same type in single varaible we need Arrays.

Some Important points —>

1. Arrays in java are treated as "Objects"
2. In java memory for objects is given in "HeapArea".
3. If memory is given in "HeapArea", jvm will give default value for the variables based on the datatype.
4. default value for boolean variable is "false".
5. **If memory is given in "StackArea", programmer should initialise the value before using the variable, otherwise it would result in "CompileTimeError".**
6. Any problem occured at runtime we categorise into 2 types

   a. Exception => Problem occurred and it can be handled.

   b. Error => Problem occured,but it can't be handled.

Note::

For every type corresponding classes are available but these classes are part of java language but not applicable at the programmer level. **These classes are also called as "proxyClasses".**

int[] [I

float[] [F

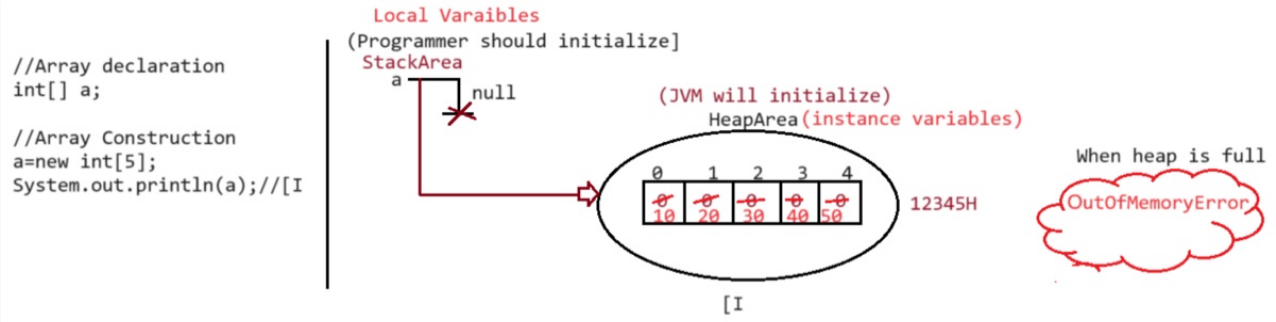
double[] [D

byte[] [B

boolean[] [Z]

short [S

String [java.lang.String...



Shortcut of way declartion,construction,initialisation in single line

int[]a = {10,20,30,40};

char[] a= {'a','e','i','o','u'};

String[] a= {"sachin","ramesh","tendulkar","IND"};

Array Element Assignements

a)- case 1:: In case of primitive array as an array element any type is allowed which can be promoted to declared type.

eg#1.

int[] a=new int[10];

a[0]=97;

a[1]='a';

byte b= 10;

a[2]=b;

short s=25;

a[3]=s;

a[4]=10L; **//CE: possible loss of precession**

b)- case 2:: In case of Object type array as an array elements we can provide either declared type object or its child class objects.

eg#1.

Object[] obj=new Object[5];

obj[0] =new Object();//valid

obj[1] =new Integer(10);//valid

obj[2] =new String("sachin");//valid

eg#2.

Number[] num=new Number[10];

num[0]=new Integer(10);//valid

num[1]=new Double(10.0);//valid

num[2]=new String("sachin")**;//invalid**

++

c)- case3:: In case of interface type array as an array element we can provide its implementation class Object.

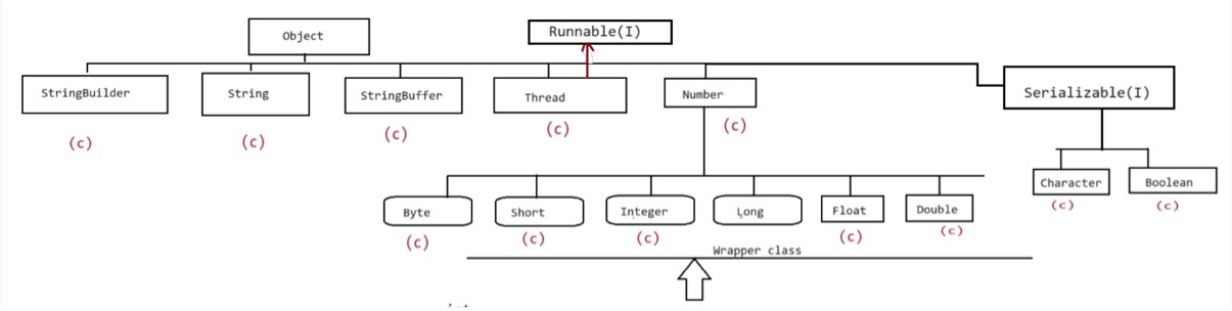
Runnable[] r=new Runnable[5];

r[0]= new Thread("sachin");

r[1]= new String("dhoni")**; //CE**

d)- case4:: In case of abstract class type array as an array element we can provide its child class object.

++



Key Differences between creating array with new keyword and without it

1)- When you use an array initializer without the new keyword, you need to provide the initial values at the time of declaration.

2)- When you use the new keyword, the array is dynamically created, and memory is allocated. The elements are initialized with default values (0 for numeric types, null for reference types).

3)- The size of an array created with the new keyword can be determined at runtime, whereas the size of an array created with an initializer is fixed at the time of declaration.

Array variable assignment

a)- case1:: Element level type promotion is not applicable

eg:: char value can be type promoted to int, but char[] can't be type promoted to int[].

int[] a= {1,2,3};

char[] c={'a','b','c'};

int[] b = a;

int[] a = c; **//invalid**

which of the following promotions are valid?

char ----> int [valid]

char[] --> int[] [invalid]

int --> long [valid]

int[] ==> long[] [invalid]

double ==> float [valid]

double[]==> float[] [invalid]

String ==> Object [valid]

String[] => Object[] [valid]

b)- case2:: In case of Object type array,its child type array can be assigned.

eg:: String[] names={"sachin","saurav","dhoni"};

Object[] obj=names;

case3:: Whenever we are assigning one array reference to another array reference,its just the reference which are being copied not the array elements.

While copying the reference only its type would be given importance,not its size.

eg:: int[] a= {10,20,30,40};

int[] b= {100,200};

a=b;

b=a;

case4:: Whenever we are copying the array,its reference will be copied but we should match it with the array dimension and its type,otherwise it would result in compile time error.

eg:: int[][] a= {{10},{20},{30}};

int[] b={100,200,300};

b= a; **//CE: incompatible type**

Note:: In array assignment,its type and dimension must be matched otherwise it would result in compile time error.

int[] a ={10,20,30};S.o.p(a); **//[I@..**

float[] f={10.0f,20.0f}; S.o.p(f); **//[F@..**

boolean[] b= {true,false}; S.o.p(b); **//[Z@..**

Integer[] i={10,20,30}; S.o.p(i); **//[L@...**

Float[] f = {10.0f,20.0f}; S.o.p(i); **//[L@...**

Working with 2D-Arrays

2D-Array = 1D-Array + 1D-Array

(ref) (data)

Declaration(All are valid)

int[][] a; **// recomended**

int a[][];

int [][]a;

int[] []a;

int[] a[];

int []a[];

ArrayConstruction

int[][] a =new int[3][2];

or

int[][] a= new int[3][];

a[0]=new int[5];

a[1]=new int[3];

a[2]=new int[1];

Array Initalisation

a[0][0] = 10;

a[2][3] = 5;

Array Declaration

int[][] a= null; **//Since 'a' is a local variable so we need to assign a value to it otherwise CE while accessing it.**

Tricky Questions

a. int[] a,b; => a-1D, b-1D

b. int a[],b[]; => a-1D, b-1D

c. int a[],b; => a-1D, b-variable

d. int a,[]b; => CE

e. int []a,[]b; => CE

f. int []a,b; => a-1D,b-1D

g. int[] a,b; // a-1D, b-1D

h. int[] a[],b; // a-2D, b-1D

i. int[] []a,b; // a-2D, b-2D

j. int[] a, []b; //CE

Note: if we want to specify the dimension before the variable, that rule is applicable only for first varaible,second varaible onwards we can't apply in the same declaration.

ShortCut to create a 2d array —>

```
int[][] b= {
    {10,20,30,40},
    {50,60}
};
```

Note: if an object does'nt have reference to access, then such objects are called as "Garbage" Object

Scanner →

```
import java.util.Scanner;
System.out.print("Enter the size of the array:");
Scanner scan= new Scanner(System.in);
size = scan.nextInt();
```

What is the difference b/w for loop and foreach loop?

forloop

=> meant for both read and write purpose.

=> it might lead to ArrayIndexOutOfBoundsException(AIOBE), if we dont' use it properly.

=> by placing the condition we can iterate from R to L and L to R.

=> Accessing of elements is through index only.

foreach loop

=> meant for only read purpose

=> The problem of exception won't occur.

=> iteration is possible only from L to R.

=> Accessing of elements is through "datatype" of the variable.

Access modifiers in java

1. private
2. public
3. protected
4. static[if we mark method as static, then that method can be called without creating the object]
5. strictfp
6. synchronized
7. final
8. abstract
9. native
10. transient
11. volatile

length property vs length() method

length: It is property which belongs to Arrays.

=> public final int length;

=> This property would return the no of elements present inside the array.

length() : It is available inside "String" class in java.

=> public int length();

=> This method would return the no of characters present in the given String.

Ananomyous Array

=> An array without a name is called Ananomyous Array.

=> These type of array is created just for instance use.

=> Creation of Ananomyous Array

eg::

```
class Test
{
```

```
    public static void main(String[] args)
    {
        System.out.println("The sum is :: "+sum(new int{10,20,30,40,50}));
    }
    static int sum(int[] arr)
    {
        int total = 0;
        for(int data : arr)
        {
            total+=data;
        }
        return total;
    }
}
```

CommandLineArguments

=> JVM will collect the arguments passed by the programmer and creates an Ananomyous array of type String.

=> JVM will call main() by passing Ananmyous array as the argument.

=> Signature of main()

public => jvm should access the main() without any authorization and authentication so make it as public.

static => jvm should not create an object of the class which contains main(), it should directly call main,so mark main() as static

void => jvm will not return anything to o.s so we need to mark the return type as void

main(String[] args) => String[] args :: it refers to command line arguments which the jvm will use to store the arguments sent by the user.

eg:: java Test 54.5 true 10

||

```
Test.main(new String{"54.5","true","10"})
public static void main(String[] args)
{
    String[] argh= {"A","B"};
    args = argh; //args = {"A","B"}
}
```