**Link: - https://www.youtube.com/playlist?list=PLd3UqWTnYXOl\_KwGTt4BZe2k3q2Yu3y1f**

**Exam Curriculum: - OCA**

**Duration: 150 minutes**

**Number of Questions : 70**

**Passing Score : 65% (Around 46 Questions you should answer)**

**Format: Multiple Choice**

**Java Basics**

* Define the scope of variables
* Define the structure of a Java class
* Create executable Java applications with a main method; run a Java program from the command line; produce console output
* Import other Java packages to make them accessible in your code
* Compare and contrast the features and components of Java such as: platform independence, object orientation, encapsulation, etc.

**Working With Java Data Types**

* Declare and initialize variables (including casting of primitive data types)
* Differentiate between object reference variables and primitive variables
* Know how to read or write to object fields
* Explain an Object's Lifecycle (creation, "dereference by reassignment" and garbage collection)
* Develop code that uses wrapper classes such as Boolean, Double, and Integer

**Using Operators and Decision Constructs**

* Use Java operators; use parentheses to override operator precedence
* Test equality between Strings and other objects using == and equals ()
* Create if and if/else and ternary constructs
* Use a switch statement

**Creating and Using Arrays**

* Declare, instantiate, initialize and use a one-dimensional array
* Declare, instantiate, initialize and use multi-dimensional arrays

**Using Loop Constructs**

* Create and use while loops
* Create and use for loops including the enhanced for loop
* Create and use do/while loops
* Compare loop constructs
* Use break and continue

**Working with Methods and Encapsulation**

* Create methods with arguments and return values; including overloaded methods
* Apply the static keyword to methods and fields
* Create and overload constructors; differentiate between default and user defined constructors
* Apply access modifiers
* Apply encapsulation principles to a class
* Determine the effect upon object references and primitive values when they are passed  into methods that change the values

**Working with Inheritance**

* Describe inheritance and its benefits
* Develop code that makes use of polymorphism; develop code that overrides methods;  differentiate between the type of a reference and the type of an object
* Determine when casting is necessary
* Use super and this to access objects and constructors
* Use abstract classes and interfaces

**Handling Exceptions**

* Differentiate among checked exceptions, unchecked exceptions, and Errors
* Create a try-catch block and determine how exceptions alter normal program flow
* Describe the advantages of Exception handling
* Create and invoke a method that throws an exception
* Recognize common exception classes (such as NullPointerException, ArithmeticException, ArrayIndexOutOfBoundsException, ClassCastException)

**Working with Selected classes from the Java API**

* Manipulate data using the StringBuilder class and its methods
* Create and manipulate Strings
* Create and manipulate calendar data using classes from java.time.LocalDateTime,  java.time.LocalDate, java.time.LocalTime, java.time.format.DateTimeFormatter, java.time.Period
* Declare and use an ArrayList of a given type
* Write a simple Lambda expression that consumes a Lambda Predicate expression

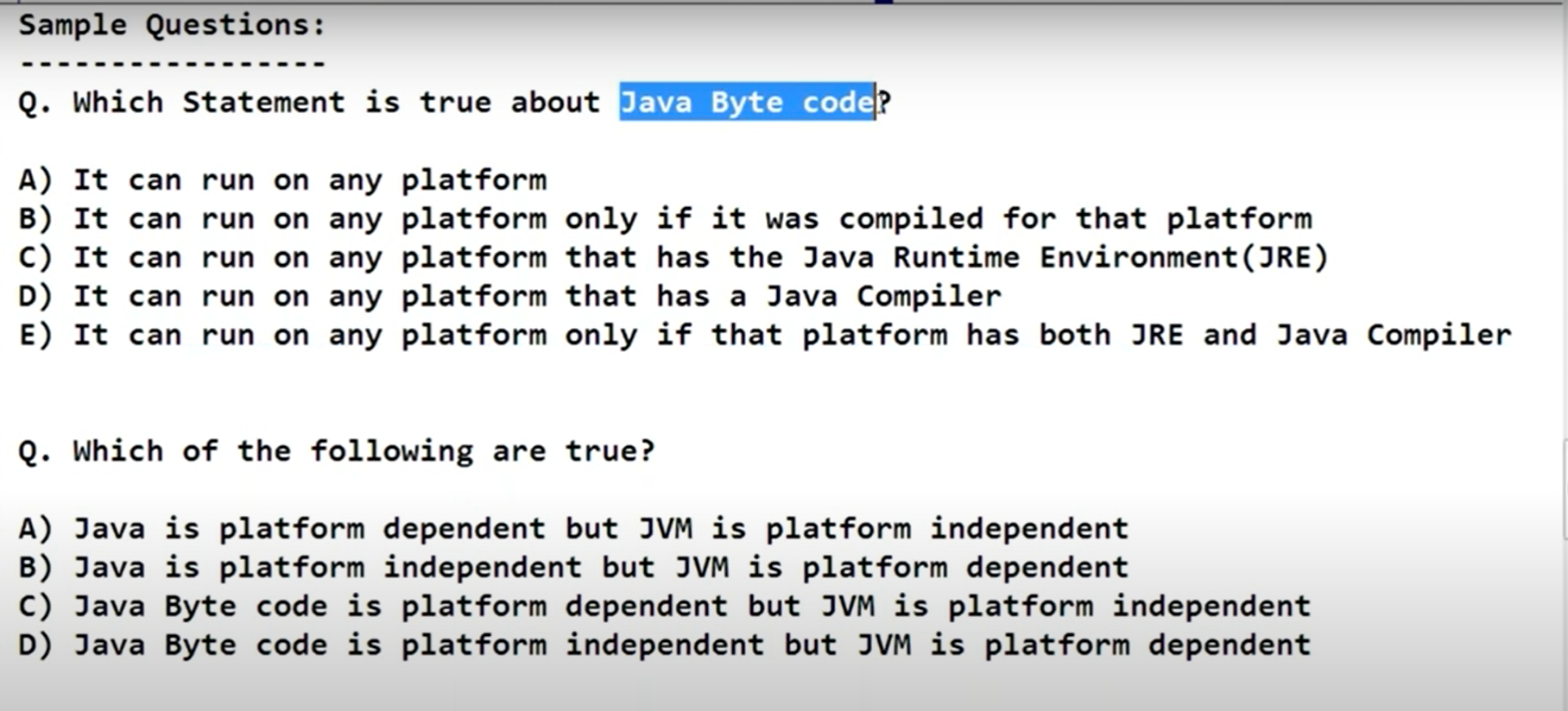
**OCJA = 1Z0-808**

**OCJP = 1Z0-809**

**Tutorial 1:**

**A computer screen shot of a computer

Description automatically generated**

****

**Test.java 🡺 Test.class (compiler) 🡺 Bytecode 🡺 Machine Code (JVM)**

**Tutorial – 2**

Java Features (Java Buzzwords)

1. Simple
2. Secure
3. Portable
4. Object Oriented
5. Robust
6. Multi Threaded
7. Architecture Neutral
8. Interpreted
9. HighPerformance
10. Distributed
11. Portable
12. Dynamic

Simple –

* Secure:

**Test.java 🡺 Test.class (compiler) 🡺 Bytecode 🡺 Machine Code (JVM)**

Inside JVM : Bytecode Verifier + Security Manager

Sandbox Checking

* Object Oriented – interms of object

Encapsulation - Security

Inheritance - Reusability

Polymorphism – Flexibility

* Robust – The chance of failing is very very rare.

Type checking: Strongly Typed PL

Garbage Collector 🡺 memory related problem is very rare

* Exception Handling:

Try

{}

catch(Exception e)

{}

* Platform Independent

Write once and run anywhere (WORA)

* Architecture Neutral

**Tutorial 3: - Nothing in it**

**Tutorial 4: -**

Buzz Words:

Platform independent – write once and run anywhere

Java Source file 🡺Test.java

Java Compiler 🡺 javac 🡺Test.java

ByteCode🡺 Test.class

Byte code is platform independent code.

**Note: .class file contains byte code and .class file obtain after compiling the .java file**

**.class file is platform independent and .class file pass to jvm to create a machine code which is platform dependent.**

**Java – platform independent**

**Byte code - platform independent**

**JVM - platform dependent**

**JDK vs JRE vs JVM**

**JDK(java development kit)**

To develop and run java applications.

**JRE (Java Runtime Environment)**

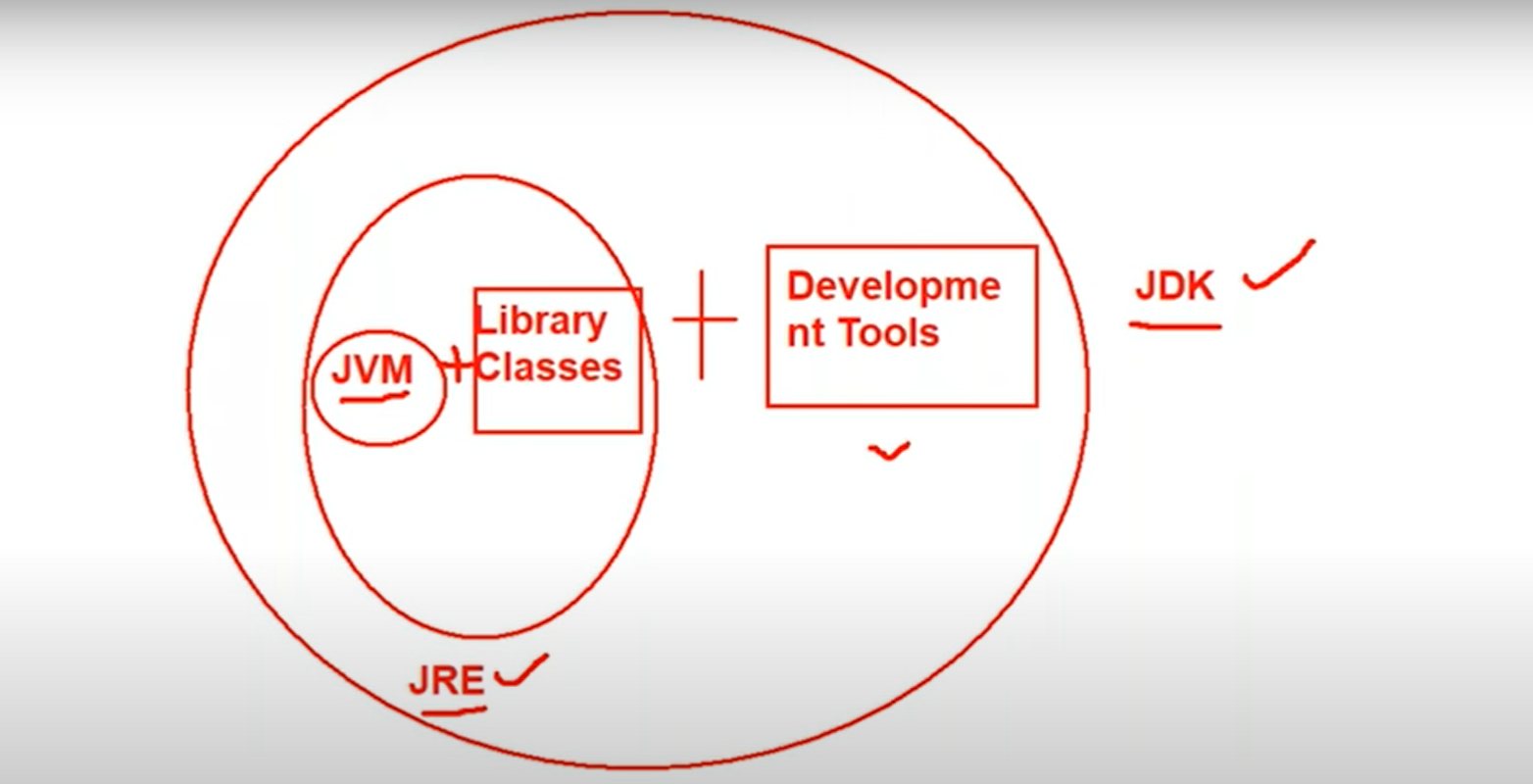
To run java applications

**JVM (Java Virtual Machine)**

Interpreter to run our java program line by line

JDK – user for development and run which contains JRE and development tools

JRE – use for just run contains JVM and library classes



A screenshot of a computer program

Description automatically generated

* 1. B and D
  2. C only

**Java Features:**

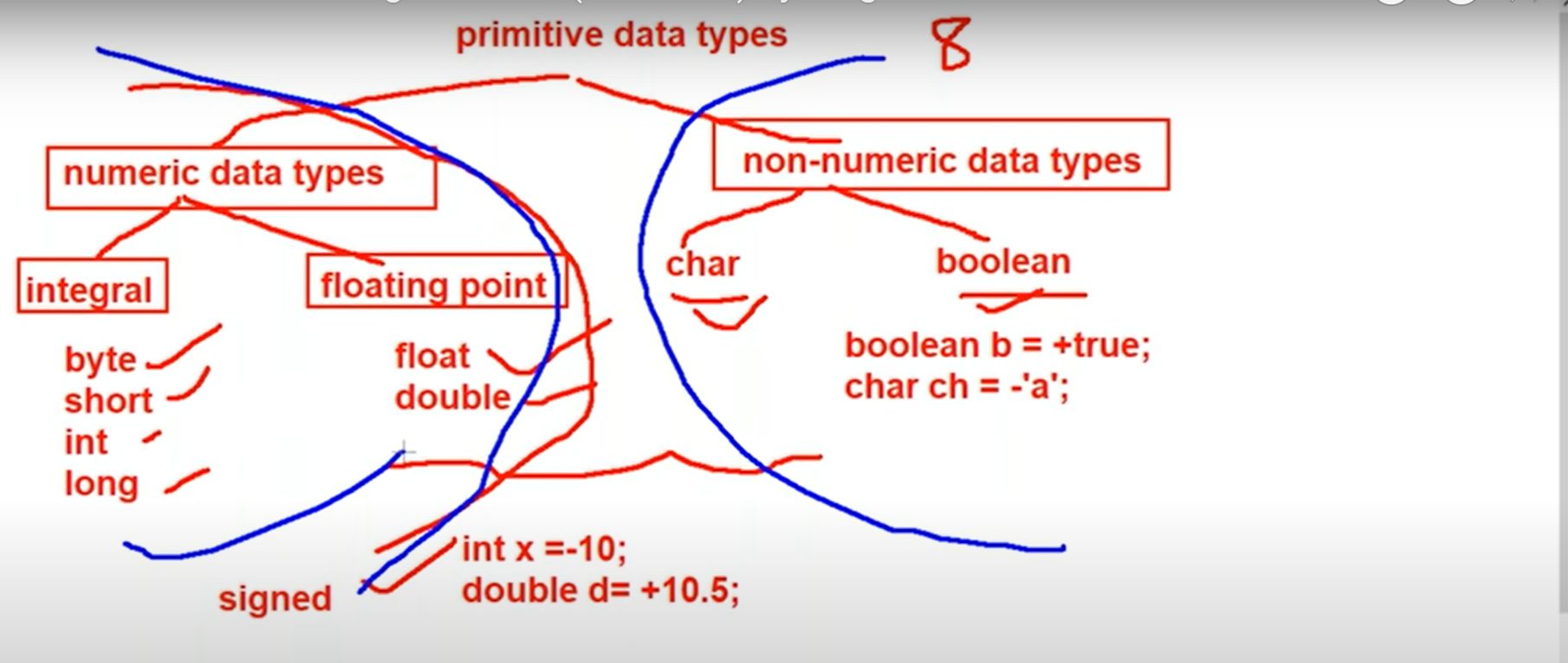
Data Types And Literals

GOTO – 31.21

Note: - Strong Type Programming Language – means it check the data type very strongly

Java is Pure OOP?

No. because it support Primitive data type



**Signed data types** = integral (byte, short, int, long) floating data types(float double)

**Numeric data types** = char Boolean

Numbers:

* 1. Integral data Types – 10,20,10000

Byte, short , int, long

* 1. Floating point types

Float and double

Non Numeric Data Types:

Char and Boolean

‘a’, ‘@’

**Tutorial -5**

Byte:

Size = 1 byte(8 bit)

+127 to -128 -128 represent in the 2’s complement form

GOTO – 48.05

Char data type

Char ch = ‘a’

Java is Unicode

>256 < 65536

Boolean data type:

True, false

**Byte = 1 byte**

**Short = 2 byte**

**Int = 4 byte**

**Long = 8 byte**

**Float = 4 byte**

**Double = 8 byte**

**Char = 2 byte**

**Boolean = NA**

**A screenshot of a computer

Description automatically generated**

**Tutorial 6:**

Literals: - constant value assigned to the variables

Int x = 10; // 10 is literals

Way to provide literal to integral data types

* 1. Decimal (base-10)
     1. to 9

int x = 12

* 1. Octal(base -8)
     1. to 7

int x = 010 –

Note: it must start with 0 – zero

* 1. Hexa Decimal (base - 16)
     1. to 9, a to f

Note: case sentivity does not matter while writing hexadecimal value

Int x = 0x10; hexadecimal value must start with 0x or 0X

Which conversion are valid

Int x = 10; - valid

Long l = 10; -valid

Long l = 10L - valid

Int x = 10L; - invalid

**Note: there is no way to assign explicitly a byte type or short type.**

**Byte b = 100b; - invalid**

**Short s = 100s; - invalid**

Whenever we assign integral literal to the byte variable under the range of byte variable it assign it the byte type or

**Floating point literals:**

By default, every floating type literal is double type (8 byte)

Float f = 123.456 - invalid

Double d = 1234.456 - valid

Float f = 123.456f - valid

Double d = 123.456F - valid

Double d = 123.456D – valid

**Note: floating point literal only accept decimal , hexadecimal and octal can’t be assigned to the floating point literals**

**In other case, integral literal , octal decimal and hexa can be assigned to integral literals**

double d = 123.456F – valid

double d = 0123.456F – invalid

double d = 0X123.456F – invalid

float f = 0XBeef ; valid because output of 0XBeef is integer and later integer could be assigned to floating literal

float f = 0XBeef.123 – invalid such combination could not proceed

Exponential form – it is always double

Double d = 1.2e3; 🡺 1.2\*10^3 🡺 1.2\*1000 🡺1200.0

Float f = 1.2e3 – invalid // exponential is always double

**Tutorial 7:**

//Float f = 1.2e3 – invalid // exponential is always double

**Note: int, short, long we can assign both value to the floating point literals i.e float and double**

Boolean:

In java Boolean value is just True and false and can’t be replaced with **1** or **0**

Boolean b = 0; - invalid // only allowed values are true or false

Sop(b);

While(1) - invalid

{

Sop(“Hello”); // Boolean can’t be replace with integer value for while loop

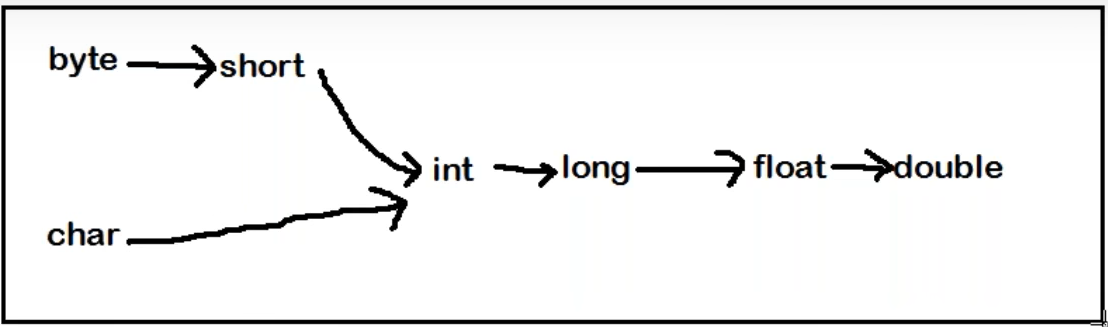
}

Char literals

Char ch = ‘a’ // must be inside single quote and only single character

\*\*\* Char ch = 97 // it will print the corresponding character till 65537

Char ch = 1972 // wil print ‘?’ because value is still not assigned to the particular Unicode



**Tutorial 8:**

//Char literals

**PRIMITIVE TYPE CASTING:**

* 1. Implicit type casting int x = ‘a’; double d = 10;
* Compiler
* Smaller data type value to bigger data type
* No loss of information
* Widening or upcasting
  1. Explicit type casting

Int x = 130;

Byte b = (byte) x ; // programmer is responsible for explicit type casting

Examples:

1. Which of the following conversion will be performed automatically in java ?

A ) int to byte - no

B) byte to int - yes

c) float to double - yes

D) double to float - not

e) None of the above

Q2. In which of the following case explicit type casting is required

1. Int to byte - yes
2. Byte to int - no
3. Float to double - no
4. Double to float - yes
5. None of the above

Q 3. Consider the code

Int I = 100;

Float f = 100.100f;

Double d = 123;

Which of the following assignment won’t compile?

1. I = f - no
2. f = I - yes
3. d = f - yes
4. f = d - no
5. d = I - yes
6. i = d - no

Q 4. In which of the following cases we will get compile time error ?

A. float f = 100F; - no

B. float f = (float)1\_11.00; - no

C. float f = 100; - no

D double d = 203.22; -

Float d= d; yes compile time error

E. int I = 100;

Float f = (float) I; no

**Tutorial 9:**

**Array:**

//double d = 10;

An array is an indexed collection of fixed numbers of homogeneous data elements

Array declaration:

All below are allowed / valid

Int [] x;

Int []x;

Int x []

Int x [];

**Int [4] x; - not allowed at the time of declaration we are not allowed to specify the size.**

**All below are valid**

Int [][] x;

Int [][]x;

Int x [][];

Int [] []x;

Int [] x[];

Int []x[];

Int [] a, b; // a- 1D. b-1D

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

Int [] [] a, b ; // a- 2D. b-2D

Int [] [] a, b [] ; // a- 2D. b-3D

**Note: if you want to specify dimension before variable this facility is only available for the first variable**

Int [] a, []b – invalid

Int [] a, b [] – valid

To get the class name

x.getclass().getName();

int [] 🡪 [I – for one dimension

int [][] 🡪 [[I for two dimensions

byte[] 🡪 [B

long [] 🡪 [L

Boolean [] 🡪 [Z

Int [] x = new int []; - invalid

Int [] x = new int [0]; - valid // array with zero size is legal - syntax wise

Class Test

{

Public static void main (String [] args)

{

}

}

Java Test A B C – **jvm is responsible for creation of string array**

**Note:**

Int x = new int [-3]; - invalid – negative value is not allowed

**import - Issue will be at the runtime or compile time ?**

**this is the issue at runtime time.**

because - new int[integer value] and -3 is integer value so compile time it will not give any error

at runtime when jvm allot a memory to array it give negative array size exception

**Tutorial 10:**

//at runtime when jvm allot a memory to array it give negative array size exception

Int []x = new int [‘a’]; // valid decleration

Int []x = new int [10L]; // invalid decleration // incompatible type … only integer value is allowed

double []x = new double [10D]; // invalid decleration // incompatible type … only integer value is allowed to declare integer

Int []x = new int [(int)10.5]; - valid

**2D Array:**

Array of Arrays concept