04-Rajdip-chavada.

MCA-02.

Java.

Date
Page

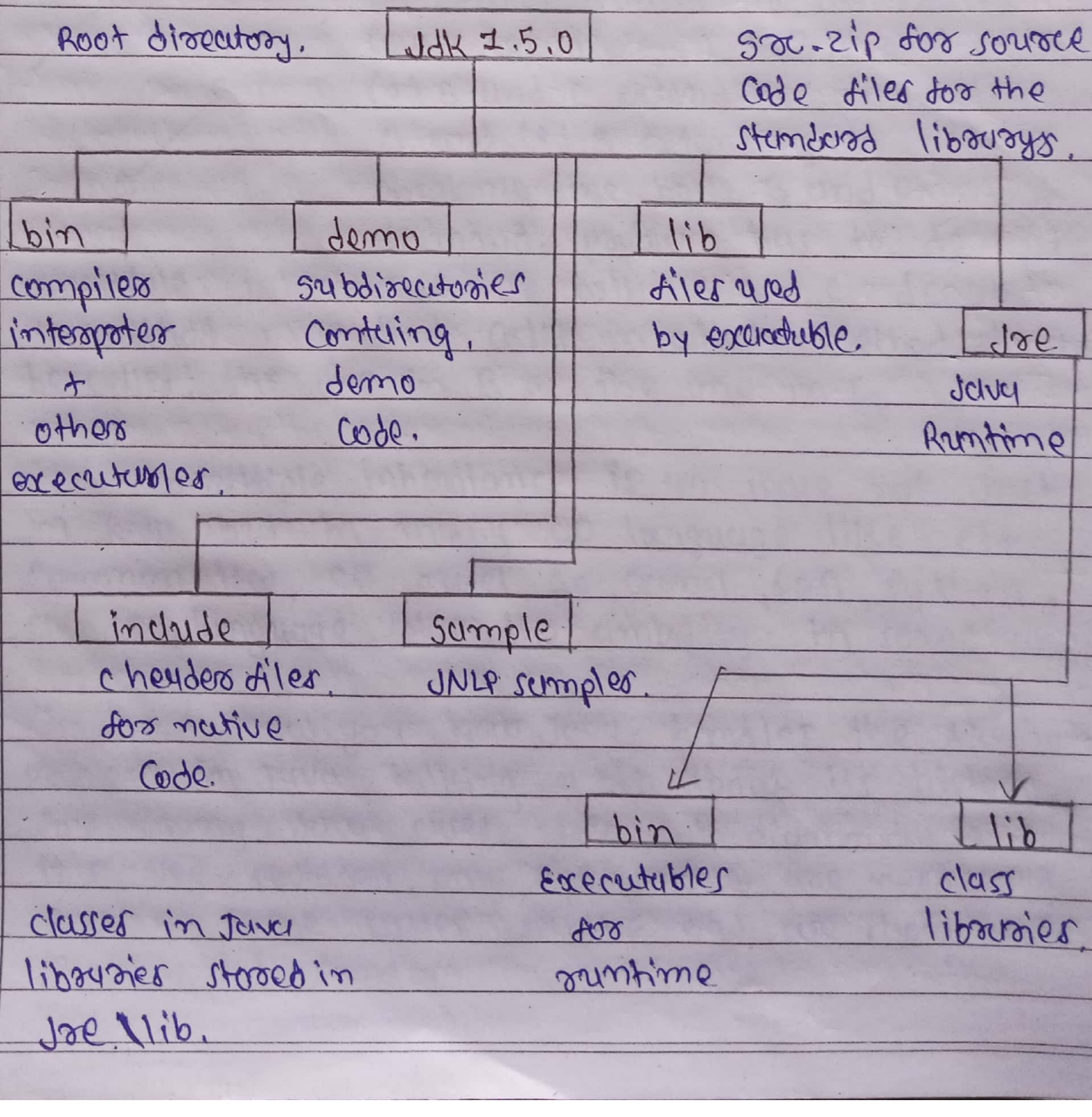
1. Explain JDK root directory with diagram.

Ans. instruction on how to install the JDK for your particular operating system from sun web sit and Oracle web site.

the JDK and the documentation are seperater, and you install them separately. the JDK for windows is available in two versions- as a web install where components are downloaded incrementally, and as a full download of an .exe file that you just execute to start execution & installation. the documentation for the JDK Consists of a large number of HTML files structured in a hierarchy that are distributed in a ZIP archive. you will find it easier to install the JDK first, followed by the documentation. If you install the JDK to drive C: under windows, the directory structure shows in figure.

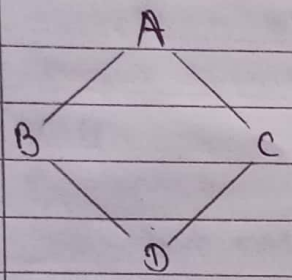
the JdK 1.5.0 directly in figure is sometimes refferd to as the root directory of Java. In some Contexts it is also refferds to as the Java home directory. the actual root directory name may have the release version number appended, in which case the actual directory name will be of the form JdK 1.5.0_n where n is a release number, so in the first maintenance release - JdK 1.5.0-01.

the Jdk 1.5.0\bin directly to the paths defined in your PATH environment variable. that way you will be able to run the Compiler and the interpreter from anywhere without having to specify the Path to it. if you installed JDK to C; then you need to add the path C:\Jdk1.5.0\bin.



Q. Discuss the diamond problem with reference to inheritance.

ans. The diamond problem is a common problem in Java when it comes to inheritance.. As simple inheritance allows a child class to derive properties from one super-class. for example, if class B inherits properties from only one super class A, then it is called inheritance and Java supports them.



It shows classes or interfaces. (Java terminology) A, B, C and D, with (1) B and C extending or implementing A and (2) D extending or implementing both B and C. the diamond problem has to do with multiple inheritance. If both B and C declare a method m and d calls m, which method should be called, the one in B or the one in C.

multiple inheritance is an issue not just in Java but in many OO language like C++, Common lisp, C#, eiffel, go, Ocaml, Perl, Python. each OO language solve the ambigity in some way.

we show you how Java handles the diamond problem in Java, talking a bit about its history, for many Java cases, it's not diamond problem, it's Vee problem. only B, C and D are needed to explain some issues. At the end, we show you a diamond problem.

1) B, C and D are classes.

the program to the right is not a legal Java Program because class D may extend only one class and here it extends both B and C.

```
class B
{
    int m()
    {   return 0; }
}
class
{
    int m()
    {
        return 1;
    }
}
class D extends B, C
{
    void P()
    {
        System.out.println(m());
    }
} // not legal in Java)
```

2) A diamond Problem

the Program to the right has the A-B-C-D diamond. we discuss variations of it. It might help to try these out in DrJava, calling method from

the Interaction Pane.

```
interface A
{
    default int m()
    {
        return 2;
    }
}
interface B extends A
{ }
interface C extends A
{ }
class D implements B, C
{
    void p()
    {
        System.out.println(m());
    }
}.
```

1) the program complies, and execution of method p in D prints 2. the call on m can be replaced by B.super.m() and C.super.m().

2) Put this method in B: default int m() { return 2; } the program remains legel. A call of method p in D prints 2 - the declaration in B overrides that in A, you can use B.super.m() in method p().

But C.super.m() won't work - it results in the error message ; bad type qualifier C in default super call method m() is overriden in B.

3. Put this method in both B and C. default int m() { return 2; }. the program is syntacically incorrect. the error message is: class D inherits unrelated default for m() from type B and C.

4. Put this abstract method in B: int m(). you get a syntax error, D does not override this abstract method.

3. Discuss finalize method and its importance in garbage Collection in Java.

Ans. This method is called automatically by Java before an object is finally destroyed and the space it occupies in memory is relesed. In practice this may be some time after in the object is inaccessible in your program. when an object goes out of scope, it is dead as far as your program is concerned, but the Java virtual machine may not get around to disposing of remains until later. its called the finalize() method for the object. the form the 'finalize() method is:

```
protected void finalize()
{
    // your code...
}.
```

garbage Collection runs sporadically in the background, it is no trivial to demonstrate it. However, one way it can be done is through the use of the finalize() method. Recall that finalize() is called when an object is about to be recycled.

– demonstrate garbage Collection via the finilize(). method.

```
class fdemo
{
        int x;
        fdemo (int i)
        {
                x=i;
        }
        // called when object is recycled.
        Protected void finalize ()
        {
                System.out.println ("finalizing" +x);
        }
        // generates an object that is immediately
          destroyed.
        void generator (int i)
        {
                fdemo  o=new fdemo (0);
        }
}
class finalize
{
        Public static void main (String args [])
```

```
{
    int count;
    fdemo ob = new fdemo (0);

    // generate a larger number of objects,
       At some Point, garbage collection will
       occur.

    for (count = 1; count < 20000; count++)
    {
        ob.generator (count);
    }
}.
```

4. define functional Programming.

ans. function programming is a programming style in which computations are coded as 'functional programming' functions. there are mathematical function like constructs. (eg. lambda functions) that are evaluated in expression contexts.

functional Programming languages are declarative, meaning that a computation's logic is expressed without describing its control flow. in declaring programming, there are no statement. Instead, programmers use expressions to tell the computers what needs to be done, but not how to accomplish the task.

A Computation in functional Programming is described by function that are evaluated in expression Contexts. these functions are not the same as the functions used in imperative programming, such as a Java method that returns a value.

5. (why was lambda expressions introduced ?) and discuss finalize method and its importance in garbage Collection and (lambda expressions & its implementation in Java).

ans. the Java language has far-reaching new language feature - the lambda expression. the impact of lambda expression will be profound, changing both the way that Programming solutions are conceptualized and how Java code is written, in the process lambda expressions can simplify & reduce the amount of source code needed to create certain Constructs, the addition of lambda expression also cause a new operator (the →) and a new syntax element to be added to the language. lambda expression help ensure that Java will remain the vibrant, nimble language that users have come to expect.

A lambda expression can implement a functional interface by defining an anonymous function that can be passed as an argument to some method... Enables support for parallel processing A lambda expression can also enables us to write parallel processing because every.

processor is a multi-core processor nowadays.

A lambda expression is, essentially, an anonyms method. However, this method is not executed on its own. Intead, it is used to implement a method defined by a functional interface. Thus, a lambda expression result in a form of anonymous class. lambda expressions are also commanly refferd to as closers.

-> lambda Expression fundamentles.

the lambda expression introduces a new syntax element and operator into the Java language. the new operator, Sometimes refferred to as the lambda operator or the arrow operator -> it devide a lambda expression into two parts. the left side specify any parameter required by the lambda expression. on the right side is the lambda bodies. one type consists of a single expression, and the other type consists of a blocks of code. we will begin with lambdas that define a single expression.

Simple type of lambda expression.

() -> 98.6

This lambda expression takes no parameter, thus the parameter list is empty. it returns the

Constraint value 98.6, the return type is inferred to be double. - (double)

```
double mymeth()
{
    return 98.6;
}
```

- the method defined by a lambda expression does not have a name.
simply the more interesting lambda expression:

```
() -> Math.random() * 100
```

the lambda expression obtains a Pseudo-random value from Math.random(), multiples it by 100, and return the result. it does not request a parameter.

Parameterise lambda expression

```
(n) -> 1.0/n.
```

-> Boolean lambda Expression - it returns the true if the value of parameter n is even and false otherwise

```
n -> (n%2)==0
```

- example

```
interface numerictest
{
        Boolean test (int n, int m);
}
class lambdademo
{
    Public static void main(String args[])
    {
        numerictest isfactor = (n,d) -> (n%d)==0;

        if (isfactor.test (20,2))
            System.out.println ("2 is a factor of 20");
        if (!isfactor.test (20,3))
            System.out.println ("3 is not a factor of 20");
        System.out.println ();

        numerictest lessthen =(n,m) -> (n<m);

        if (lessthen.test (2,10))
            System.out.println (" 2 is less than 20");
        if(!lessthen.test (20,2))
            System.out.println (" 20 is not less then 2 ");
        System.out.println ();

        numerictest absEqual =(n,m) -> (n<0 ? -n : n)
                    == (m<0 ? -m : m);

        if (absEqual.test (4,-4))
            System.out.println (" Absolute values of 4
                    and -4 are equal. ");
```

```
if (!lessthan.test(4,-5))
        System.out.println("Absolute value of 4
                and -5 are not equal.");
    System.out.println();
  }
}.
```

— Output

2 is a factor of 10
3 is not a factor of 10

2 is less then 10
10 is not less than 2

Absolute values of 4 and -4 are equal.
Absolute values of 4 and -5 are not equal.

6. Discuss the benefit of generic over non-generic type.

ans. the term generics means parameterized types. parameterized types are important because they enable you to create classes, interface, and methods in which the type of data upon which they operate is specified as a parameter. A class, interface, or method that operates on a type parameter is called generic, as in generic class or generic method.

- Stronger type checks at Comple time. a Java compiler applies strong type checking to generic codes and issues errors if the code violates type safety...

- elimination of casts..
- enabling programmers to implement generic algorithms.

- Programs that uses generics has got many benefits over non-generic code.

1. Code Reuse - by using generics, one needs to write a method/class/interface only once and use for any type. whereas in the non-generics, the code needs to be written again and again whenever needed.

2. type safety - generics make errors to appear compile time than at run time. (it's always better to know problems in your code at compile time rather than making your code fail at run time.).

example - to create an Arraylist that store name of students and if by mistake programmer adds an integer object instead of string, Compiler allow it. when this data is retrieved from Arraylist, it causes problems at runtime for non-generic Arraylist.

```
import Java.util.*;
class test
{
        Public static void main(String args[])
        {
```

```
Arraylist al = new Arraylist ();
al.add ("Jax");
al.add ("Rajbip");
al.add (20);

String s1 = (String) a1.get (0);
String s2 = (String) a2.get (2);

try
{
    String s3 = (String) at.get (2);
}
catch (Exception e)
{
    System.out.println ("Exception : " + e);
}
    }
}
```

- Output

Exception :
Java.lang.ClassCastException :
Java.lang.Integer cannot be cast to Java.lang.String.

- How generics solve this problem.

If this list was made generic, then it would take only String objects & threw compile time error in any other case.

```
import Java.util.*;
class test
{
```

```
Public static void main (String args[])
{
    arraylist <String> al = new arraylist <String>();

    al.add ("Sachin");
    al.add ("Rahul");
    al.add (20); // no suitable method found.
    String s1 = al.get (0);
    String s2 = al.get (1);
    String s3 = al.get (2);
}
}
```

- Solve by gemric

```
import java.util.*;
class test
{
    Public static void main (String args[])
    {
        arraylist <String> al = new arraylist <String> ();
        al.add ("dalx");
        al.add ("Rahul");
        String s1 = al.get (0);
        String s2 = al.get (1);
    }
}.
```

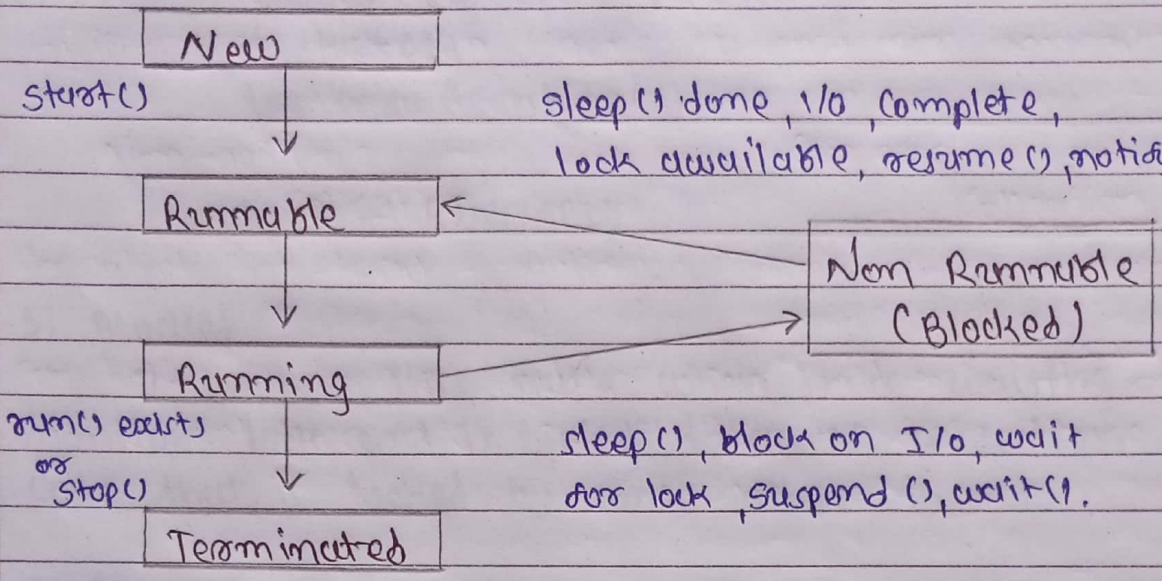
There are the some points, which will describe difference between generics and non generics.

Non-generic Collection	Generic Collection
• Arraylist list = new Arraylist();	• Arraylist list = new Arraylist();
• Can hold any type of datas. Hence not type safe	• Can hold only the defined type of datas. Hence type safe.
• Individual type casting needs to be done at every retrieval.	• No type casting is needed.
• checked for type safety at runtime	• check for type safety at Compile-time.

7. Explain thread life-cycle with diagram.

ans. A thread can be in one of the five states. According to sun, there is only 4 state in thread life cycle in Java new, runnable, non-runnable and terminated. there is no running state.

But for better understanding the threads explaind in 4 or 5 states. the life cycle of the thread in Java is controlled by JVM. the Java thread states are the new, runnable, running, blocked (non-runnable) and terminated.



1) New

A new thread (or a Born thread) is a thread that's been created but not yet started. it remains in this state until we start it using the start() method.

```
Runnable runnable = new
Newstate();
Thread t = new
Thread (runnable);
Log.info (t.getstate());
```

the method t.getstate() prints New.

2) Runnable

Threads in this state are either running or ready to run, but they're waiting for resource allocation from the system.

```
Runnable runnable =new
Newstate();
Thread t=new
```

```
Thread (runnable);
t.start();
log.info(t.getstate());
```

output - RUNNABLE

3) Blocked

It enters this state when it is waiting for a monitor lock and is trying to access a section of code that is locked by some other thread.

```
Public class blockedstate
{
    Public static void main (String args[]) throws
    InterruptedException
    {
        Thread t2 = new Thread (new demothreadB());
        Thread t2 = new Thread (new demothreadB());
        t2.start();
        t2.start();
        Thread.sleep(2000);
        log.info(t2.getstate());
        System.exit();
    }
}
class demothreadB implements Runnable.
{
    @Override
    Public void run()
    {
        CommonResource();
    }.
```

```
Public static synchronized void commonResource()
{
    while (true)
    {
        // infinite loop to mimic heavy processing
        // t2 won't leave this method
        // when t2 try to enter this
    }
}
}.
```

Output → BLOCKED.

4) Terminated.

It's in the terminated state when it has either finished execution or was terminated abnormally.

```
Public class terminatedState implements Runnable
{
    Public static void main (String args[]) throws
    InterruptedException
    {
        thread t2=new thread(new terminatedState());
        t2.start();
        thread.sleep(2000);
        log.info(t2.getState());
    }
    @Override
    Public void run()
    {
        // No Procesing in this block
    }
```

}

output – TERMINATED.

A thread is alive if and only if it has been started and has not yet died.

8. Discuss Application Thread with example.

ans

An application consists of one or more processes... one or more threads run in the context of the process. A thread is the basic unit to which the operating system allocates processor time. A thread can execute any part of the process code, including parts currently being executed by another thread. Java provides integrated support for multithread programming.

A multithreaded program contains two or more parts that can run concurrently, each part of such a program is called a thread & each thread defines a separate path of execution. multithreading is a specialized form of multitasking.

Multithreading in Java is a process of executing two or more threads simultaneously to maximum utilization of CPU. multithreaded applications execute two or more threads run concurrently. Hence, it is also known as Concurrently. in Java, each thread runs parallel to each other. also context switching between threads takes less time.

```
Package demotest;
Public class guruthread implements Runnable.
{
    Public static void main (String args[])
    {
        Thread guruthread1 = new Thread ("T1");
        Thread guruthread2 = new Thread ("T2");
        guruthread1.start();
        guruthread2.start();

        System.out.println (" thread names are following: ")
        System.out.println (" guruthread1.getname());
        System.out.println ("guruthread2.getname());
    }
    @Override
    Public void run()
    {
    }
}.
```

- Advantages of Multithreading (Application of threading).
  - the users are not blocked because threads are independant, and we can perform multiple operations at times.
  - As such the threads are independant, the others threads won't get affected if one thread meets an exception.

9. Difference between "==" and equals() in Java.

ans. The equals() method compares the character sequence of two string objects for equality. Applying the == to two string references simply determines whether the two references refer to the same object.

In Java both equals() and "==" operator in Java are used to compare objects to check equality but here are some of the differences between the two:

1. The main difference is that one is a method and the other is the operation
2. we use == operators for references comparison (address) Comparison) and .equals() method for content Comparison. and "==" checks if both objects point to the same memory location whereas .equals() evaluates to the comparison of values in the objects.
3. If a class does not override the equals method, then by default it uses the equals (object o) method of the closest parent class that has overridden this method.

example

```
Public class test
{
    Public static void main (string args[])
    {
```

```
String s1 = "Hello";
String s2 = "Rajdip";  ="Hello";
String s3 = new String ("Hello");

System.out.println (s1==s2);
System.out.println (s1==s3);
System.out.println (s1.equals(s2));
System.out.println (s1.equals (s3));
    }
}
```

Output

true
false
true
true

20 Discuss any five byte stream classes.

ans Byte streams are defined by using two class hierarchies. at the top of these are two subabstract classes.

InputStream
OutputStream

defines the characteristics common to byte input streams and outputStream

from InputStream & outputstream are created several concrete subclasses that offers varying functionality & handle the detail of reading of & writing to various devices. ,such as disk files.

- InputStream - top level abstract class for byte-oriented input stream.

- ByteArrayInputStream - An instance of this class contains an internal buffer to read bytes stream.

- FilterInputstream - An instance of this class contains some other input stream as a basic source of data for further manipulation.

- OutputStream - top-level abstract class for byte-oriented input stream.

- ByteArrayOutputStream - An instance of this class contains an internal buffer to write a bytes stream.

Example.

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

Public class Bytes.
{
    Public static void main (string args[]) throws
     IOException
    {
        fileInputstream in=null;
        FileInputstream out=null;
```

```
try
{
    in = new FileInputStream ("new.txt");
    out = new FileInputStream ("out.txt");
    int c;
    while ((c = in.read()) != -1)
    {
        out.write(c);
    }
}
finally
{
    if(in != null)
    {
        in.close();
    }
    if (out != null)
    {
        out.close();
    }
}
}
}.
```

11. Discuss situations where String tokenizer class is used.

ans. the string tokenizer class allows an application to breaks a string into tokens. the tokenization method is much simpler than the one used by the streamtokenizer class. the stringtokenizer methods do not distinguish among identifiers.

numbers, and quoted strings, nor do they recognize and skip comments.

The set of delimiters may be specified either at creation time or on a per-token basic.

An instance of StringTokenizer behaves in one of two ways, depending on whether it was created with the returnDelims flag having the value true or false.

- if the flag is false, delimeter characters serve to separate tokens. A token is a maximal sequence of consecutive characters that are not delimiters.

- if the flag is true, delimeter characters are themselves considered to be tokens. A token is thus either one delimeter character, or a maximal sequence of consecutive characters that are not delimiters.

A StringTokenizer object internally maintains a current position within the string to be tokenized. Some operations advance this current position past the characters processed.

example

```
StringTokenizer st = new StringTokenizer("this is a
test");
while (st.hasMoreTokens())
{
    System.out.println(st.nextToken());
}
```

output.

this
is
a
test.

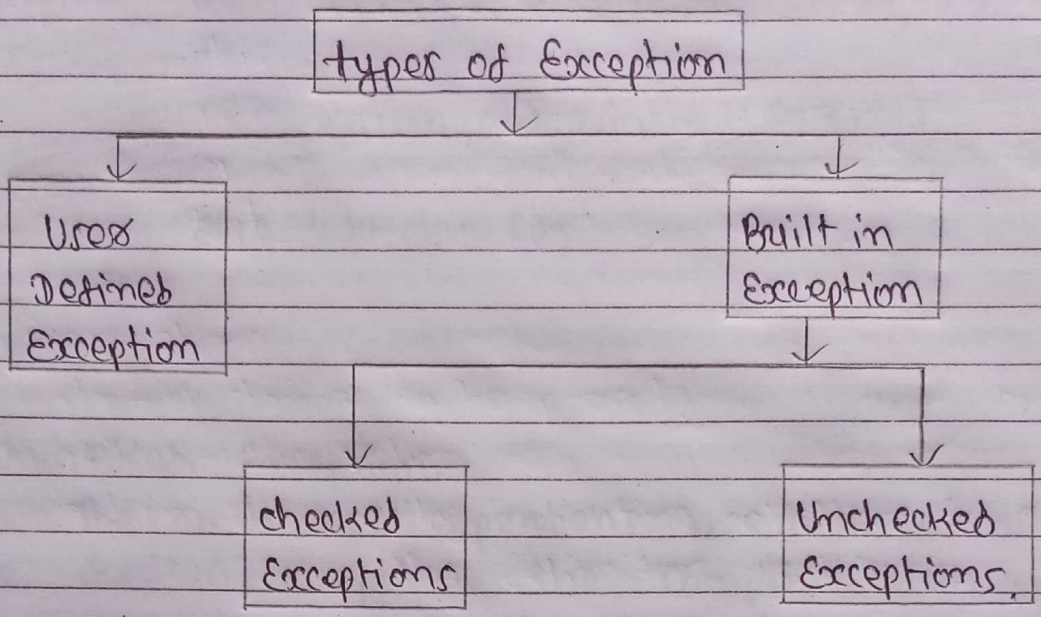
→ Parameters.
str - a string to be parsed.
delim - the delimiters.
returndelims - flag indicating whether to return the delimiters as tokens.

→ Throws
NullPointerException - if str is null.

12. Discuss Exception and types of Exceptions in Java.

ans. exception is an event that occurs during the execution of an program and disrupts the normal flow of the program's instructions. Bugs or errors that we don't want and restrict our program's normal execution of code are referred to as exceptions.

Exceptions that are already available in Java libraries are refered to as built-in exception. these exceptions are able to define the error situation so that we can understand the reason of getting this error. it can be categorized into two broad categories.



- checked Exception

checked exceptions are called Compiletime exceptions because these exceptions are checked at Compile-time by the Compiler. the Compiler ensures. whether the programmer handles the exception or not. the programmer should have to handle the exception; otherwise, the system has show a Compilation error.

ex

```
import java.io.*;
class checkedExceptionExample
{
    Public static void main (string args[])
    {
        Fileinputstream file_data = null;
        file_data = new FileInputstream ("Hi.txt");
        int m;
```

```
while ((m = file_data.read()) != -1)
{
    System.out.println((char)m);
}
file_data.close();
}
}
```

- Unchecked Exception.

The unchecked exceptions are just opposite to the checked exception. the compiler will not check these exceptions at compile time. in simple words, if a Program throws an unchecked exception., and even if we didn't handle or declare it, the program would not give a compilation error. usually, it occurs when the users provids bad data during the interaction with the program.

ex

```
import java.util.*;
class UserdefinedException
{
    Public static void main (String args[])
    {
        try
        {
            throw newException (5);
        }
        Catch (NewException exc)
        {
            System.out.println (exc);
        }
    }
}
```

```
class NewException extends Exception
{
    int x;
    NewException (int y)
    {
        x = y;
    }
    Public String toString ()
    {
        return ("Exception value = " + x);
    }
}
```

output

Exception value = 5.

there are two types of Exception in Java programming Language.

13. Discuss three subclasses of Error Runtime Exception.

ans RuntimeException and its subclasses are unchecked exceptions. unchecked exceptions do not need to be declared in a method or Constructor's throws clause if they can be throws by the execution of the method or Constructor & Propagate outside the method or constructor boundary.

ArithmeticException

this exception is thrown to indicate an exceptional arithmetic condition, such as integer division by zero.

ArrayIndexOutOfBoundsException

this exception is thrown when an out-of-range index is detected by an array object. an out-of-range index occurs when the index is less than zero or greater than or equal to the size of the array.

ArrayStoreException

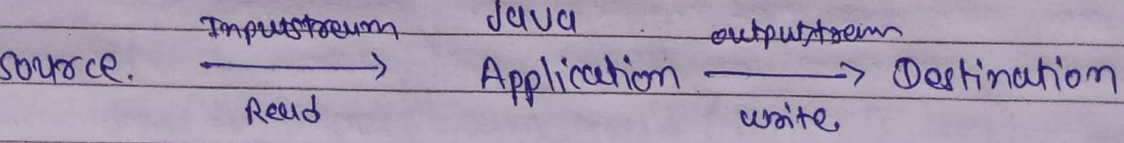
this exception is thrown when there is an attempt to store a value in an array element that is incompatible with the type of the array.

ClassCastException

this exception is thrown when there is an attempt to cast a reference to an object to an inappropriate type.

14. Explain seven different subclasses of InputStream.

ans. InputStream class in the superclass of all the io classes, representing an input stream of bytes. It represents input stream of Bytes. Applications that are defining subclasses of InputStream must provide method, returning the next byte of input. A reset() method is invoked which repositions the stream to the recently marked position.



1) mark()

Java.io.inputstream.mark(int arg) marks the current position of the input stream. it sets readlimit

Syntax -

Public void mark(int arg)

2) Read()

Java.io.Inputstream.read() reads next byte of data from the input stream. the value byte is returned in the range 0 to 255. if no byte is available because the end of the stream has been reached, the value -1 is returned.

Public abstract int read()
- Return - Reads next data else, -1.
- Exception - IOException - if I/O error occurs.

3) close()

Java.io.InputStream.close() closes the input stream and releases system resources associated with this stream to Garbage Collector.

Public void close()
- Return - void
- Exception - If I/O error occurs.

4) read()

Java.io.InputStream.read(byte[] arg) reads number of bytes of arg.length from the input.

stream to the buffer array arg. the bytes read by read() method are returned as int. If len is zero, then no bytes are read and 0 is returned.

Public int read(byte[] arg)
- Parameter - array whose number of bytes to be read.
- Return - reads number of bytes & return to the buffers else, -1.
- Exception - IOException - If I/o error occur
  - NullPointerException - if arg is null.

5) reset()

Java.io.InputStream.reset() is invoked by mark() method. it repositions the input stream to the marked position.

Public void reset()
- Return - void
- Exception - IOException - if I/o error occurs.

6) markSupported()

Java.io.InputStream.markSupported() method tests if this input stream supports the mark and reset methods. the markSupported method of InputStream returns false by default.

Public boolean markSupported()
- Return - true if inputstream supports the mark() and reset() method else, false.

7) skip()

Java.io.Inputstream.skip(long arg) skips and discards arg bytes in the input stream.

Public long skip(long arg)

- arg - no. of bytes to be skipped
- Return - skip bytes.
- Exception - IOException - if I/o error occurs.

15. Explain channels with reference to file-handling in Java.

ans. Chanels are defind in java.io.channels & the channels are algorithms to streams in the original I/o Package. A channel represents an open connection to an input/output device, such as a file or a socket.

A Java NIO FileChannel is a channel that is connected to a file. using a file channel you can read data a file. and write data to a file. the Java NIO FileChannel class is NIO an alternative to reading files with the standard Java IO API.

A FileChannel cannot be set into non-blocking mode. it always runs in blocking mode.

- Opening FileChannel.

```
RandomAccessFile a-File = new RandomAccessFile
                          ("new.txt", "rw");
file channel in = aFile.getchannel();
```

- Reading data from filechannel.
  - to read data from file firstly you can your read() method. first a buffer is allowed. the data read from the filechannel is read into Buffer. and second filechannel.read() method is called.

```
ByteBuffer B = ByteBuffer.allocate(48);
int byteRead = inchannel.read(B);
```

- writing data ~~for~~ to filechannel
  - to write data into file - filechannel.write() method is used.

```
String newdata = "Hello " + System.currentTimeMillis();
ByteBuffer B = ByteBuffer.allocate(48);
B.clear();
B.put(newdata.getByte());
B.flip();
while (buf.hasRemaining())
{
        channel.write(buf);
}
```

- closing a filechannel.

```
channel.close();
```

- filechannel position - long pos channel.position();

```
channel.position(pos + 1231);
```

- size and truncate

```
- long filesize = channel.size();
- channel.truncate(2024);
```

16. Explain Synchronization & Deadlock in threading.

ans. when using multiple threads, it is sometimes necessary to coordinate the activities of two or more. the process by which this is achieved is called Synchronization, the most common reason for synchronization is when two or more threads need access to a shared resource that can be used by only one thread at a time.

Key to synchronization in Java is the concept of the monitor, which controls access to an object. A monitor works by implementing the concept of a lock. when a object is locked by one thread, no other thread can gain access to the object. when the thread exists, the object is unlocked & is available for use by another thread.

Synchronization is supported by the keyword synchronization & a few well-defined methods that all objects have. since synchronization was designed into Java from the synchronization of objects is almost transparent.

example

```
class sumarray
{
    private int sum;
    synchronized int sumarray (int nums[])
    {
        sum=0
        for (int i=0; i<nums.length; i++)
        {
```

```
            sum += nums[i];
            System.out.println("Running total for " +
                Thread.currentThread().getName() +
                " is " + sum);
            try
            {
                Thread.sleep(10);
            }
            catch (InterruptedException exc)
            {
                System.out.println("Thread interrupted.");
            }
        }
        return sum;
    }
}
class mythread implements Runnable
{
    Thread thrd;
    static sumarray su = new sumarray();
    int a[];
    int answer;

    mythread(String name, int nums[])
    {
        thrd = new Thread(this, name);
        a = nums;
        thrd.start();
    }
    Public void run()
    {
        int sum;
```

```
        System.out.println(thrd.getName() + "
                                    starting.");
        answer = su.sumarray(a);
        System.out.println("sum for " + thrd.getName
            () + " is " + answer);
        System.out.println(thrd.getName() + "
                                    terminating.");
    }
}
class Sync
{
    Public static void main (String args[])
    {
        int a[] = {1,2,3,4,5};
        mythread m1 = new mythread ("child #1", a);
        mythread m2 = new mythread ("child #2", a);

        try
        {
            m1.thrd.join();
            m2.thrd.Join();
        }
        catch(InterruptedException exce)
        {
            System.out.println("main thread
                                interrupted.");
        }
    }
}
```

Output.

```
child #1 starting.
Running total for child #1 is 1
child #2 starting.
Running total for chid #1 is 3
Running total for chil8 #1 is 6
Running total for child #1 is 10
Running total for chil8 #1 is 15
sum for child #1 is 15.
child #1 terminating.
Running total for child #2 is 1
Running total for chil8 #2 is 3
Running total for child #2 is 6
Running total for child #2 is 10
Running total for child #2 is 15
sum of child #2 is 15
child #2 terminating.
```

Deadlock.

Deadlock is, as the name implies, a situation in which one thread is waiting for another thread to do something, but that other thread is waiting on the first, thus, both threads are suspendent waiting on each other, & neither executs, this situation is analogous to two overly polite.

eg

```
Public class test
{
    Public static void main (String args[])
    {
        final String resource1 = "Raj";
```

```
final string resource2 = "dip";
Thread t2 = new Thread()
{
    Public void run()
    {
        Synchronized (resource1)
        {
            System.out.println("Thread 1:
                locked resource 1");
            try
            {
                Thread.sleep(200); } catch (Exception e){}

            Synchronized (resource2)
            {
                System.out.println("Thread 1: locked
                resource 2");
            }
        }
    }
};
Thread t2 = new Thread() {
    Public void run(){
    synchronized (resource2) {
        System.out.println("Thread 2: locked resource 2");
    try {Thread.sleep(100);} catch (Exception e) {}
    synchronized (resource1) {
        System.out.println("Thread 2: locked resource 1");
    }
}
};
t1.start();
t2.start();
}
}
```

output

Thread 1 : locked resource 1
Thread 2 : locked resource 2.