**Core java Task**

DATE :- 5/1/2018

**1.Write 3 different java programs to print the following patterns**

**a) 1**

**12**

**123**

**12345**

import java.io.\*;

// Java code to demonstrate number pattern

public class Pattern1

{

    // Function to demonstrate printing pattern

    public static void printNums(int n)

    {

        int i, j,num;

        // outer loop to handle number of rows

        //  n in this case

        for(i=0; i<n; i++)

        {

            // initialising starting number

            num=1;

            //  inner loop to handle number of columns

            //  values changing acc. to outer loop

            for(j=0; j<=i; j++)

            {

                // printing num with a space

                System.out.print(num+ " ");

                //incrementing value of num

                num++;

            }

            // ending line after each row

            System.out.println();

        }

    }

    // Driver Function

    public static void main(String args[])

    {

        int n = 5;

        printNums(n);

    }

}

**b) 54321**

**5432**

**543**

**54**

**5**

import java.io.\*;

// Java code to demonstrate number pattern

public class Pattern1

{

    // Function to demonstrate printing pattern

    public static void printNums(int n)

    {

        int i, j;

        // outer loop to handle number of rows

        //  n in this case

        for(i=0; i<n; i++)

        {

            //  inner loop to handle number of columns

            //  values changing acc. to outer loop

            for(j=n; j>=i; j--)

            {

                // printing num with a space

                System.out.print(j+” ”);

            }

            // ending line after each row

            System.out.println();

        }

    }

    // Driver Function

    public static void main(String args[])

    {

        int n = 5;

        printNums(n);

    }

}

**c)     x**

**xxx**

**xxxxx**

**xxxxxxx**

**xxxxx**

**xxx**

**x**

**Note: Shape will be Rhombus.**

  import java.util.Scanner.\*;

public class Diamond

{

Static Void display(int n,char c)

{

int i=1;

int j;

while(i<=n)

{

j=1;

while(j++<=n-i)

{

System.out.print(" ");

}

j=1;

while(j++<=i\*2-1)

{

System.out.print(c);

}

System.out.println();

i++;

}

i=n-1;

while(i>0)

{

j=1;

while(j++<=n-i)

{

System.out.print(" ");

}

j=1;

while(j++<=i\*2-1)

{

System.out.print(c);

}

System.out.println();

i--;

}

}

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter N : ");

int n=sc.nextInt();

System.out.print("Enter Symbol : ");

char c = sc.next().charAt(0);

Diamond.display(n,c);

}

}

**2. Write a java program to take the input from user and determine if it is a prime number or not.**

import java.util.Scanner;

public class JavaProgram

{

public static void main(String args[])

{

int num, i, count=0;

Scanner scan = new Scanner(System.in);

System.out.print("Enter a Number : ");

num = scan.nextInt();

for(i=2; i<num; i++)

{

if(num%i == 0)

{

count++;

break;

}

}

if(count == 0)

{

System.out.print("This is a Prime Number");

}

else

{

System.out.print("This is not a Prime Number");

}

}

}

**3. Write a java program to display the fibonacci series till less than 200 using only 2 variables.**

public class fibonacci {

public static void main(String args[]){

int febCount =Integer.parseInt(args[0]);

int[] feb = new int[febCount];

int c=0;

feb[0] = 0;

feb[1] = 1;

for(int i=2; i < febCount; i++){

c++;

feb[i] = feb[i-1] + feb[i-2];

if(feb[i]>=200)

{

break;

}

}

for(int i=0; i<=c; i++){

System.out.print(feb[i] + " ");

}

}

}

**5.Write Java program to check if a name is palindrome.**

**import java.util.\*;**

class Palindrome

{

public static void main(String args[])

{

String original, reverse = ""; // Objects of String class

Scanner in = new Scanner(System.in);

System.out.println("Enter a string to check if it is a palindrome");

original = in.nextLine();

int length = original.length();

for ( int i = length - 1; i >= 0; i-- )

reverse = reverse + original.charAt(i);

if (original.equals(reverse))

System.out.println("Entered string is a palindrome.");

else

System.out.println("Entered string is not a palindrome.");

}

}

**6.Write Java program to check if a number is Armstrong number or not? (input 153 output true,  123 output false)**

  public class Armstrong {

public static void main(String[] args) {

int number, originalNumber, remainder, result = 0, n = 0;

number=Integer.parseInt(args[0]);

originalNumber = number;

for (;originalNumber != 0; originalNumber /= 10, ++n);

originalNumber = number;

for (;originalNumber != 0; originalNumber /= 10)

{

remainder = originalNumber % 10;

result += Math.pow(remainder, n);

}

if(result == number)

System.out.println("true");

else

System.out.println("false");

}

}

**7.How to find factorial of number in Java using iteration?**

public class Factorial {  
  
   public static void main(String[] args) {  
       int n = 7;  
       int result = 1;  
       for (int i = 1; i <= n; i++) {  
           result = result \* i;  
       }  
       System.out.println("The factorial of 7 is " + result);  
   }  
}

**8.Write a Java code to take a character as a input from user and determine if it is a vowel or a consonant using conditional construct.**

class Char

{

public static void main(String[ ] arg)

{

int i=0;

Scanner sc=new Scanner(System.in);

System.out.println("Enter a character : ");

char ch=sc.next( ).charAt(0);

//char ch=sc.nextChar();

switch(ch)

{

case 'a' :

case 'e' :

case 'i' :

case 'o' :

case 'u' :

case 'A' :

case 'E' :

case 'I' :

case 'O' :

case 'U' :i++;

}

if(i==1)

System.out.println("Entered character "+ch+" is Vowel");

else

if((ch>='a'&&ch<='z')||(ch>='A'&&ch<='Z'))

System.out.println("Entered character "+ch+" is Consonent");

else

System.out.println("Not an alphabet");

}

}

**9. Write a switch case java code to create calculator with + - / \* functionalities only.**

import java.util.Scanner;

class calc

{

int sum;

float d;

void add(int a,int b)

{

sum=a+b;

System.out.println("sum => "+sum);

}

void sub(int a,int b)

{

sum=a-b;

System.out.println("sum => "+sum);

}

void mul(int a,int b)

{

sum=a\*b;

System.out.println("sum => "+sum);

}

void div(float v1,float v2)

{

d=v1/v2;

}

public static void main(String args[])

{

char ch,c;

int x,y;

float f1,f2;

Scanner sc=new Scanner(System.in);

calc cc=new calc();

do{

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*calc\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("+ for Addition");

System.out.println("- for Addition");

System.out.println("/ for Addition");

System.out.println("\* for Addition");

System.out.println("Enter your choice => ");

ch=sc.next().charAt(0);

switch(ch)

{

case '+':

System.out.println("Enter values for x= > ");

x=sc.nextInt();

System.out.println("Enter values for y= > ");

y=sc.nextInt();

cc.add(x,y);

break;

case '-':

System.out.println("Enter values for x= > ");

x=sc.nextInt();

System.out.println("Enter values for y= > ");

y=sc.nextInt();

cc.sub(x,y);

break;

case '\*':

System.out.println("Enter values for x= > ");

x=sc.nextInt();

System.out.println("Enter values for y= > ");

y=sc.nextInt();

cc.mul(x,y);

break;

case '/':

System.out.println("Enter values for f1= > ");

f1=sc.nextInt();

System.out.println("Enter values for f2= > ");

f2=sc.nextInt();

cc.div(f1,f2);

break;

default:

System.out.println("Invalid choice");

break;

}

System.out.println("Do you want to continue[n/y]....");

c=sc.next().charAt(0);

}while(c!='n');

}

}

**10. Write a java code to copy one array into another.**

public class Test

{

public static void main(String[] args)

{

int a[] = {1, 8, 3};

// Create an array b[] of same size as a[]

int b[] = new int[a.length];

// Doesn't copy elements of a[] to b[], only makes

// b refer to same location

b = a;

// Change to b[] will also reflect in a[] as 'a' and

// 'b' refer to same location.

b[0]++;

System.out.println("Contents of a[] ");

for (int i=0; i<a.length; i++)

System.out.print(a[i] + " ");

System.out.println("\n\nContents of b[] ");

for (int i=0; i<b.length; i++)

System.out.print(b[i] + " ");

}

}

11. Write a java code to compare the length of two arrays and display the longer array.

import java.util.Arrays;

class Test

{

public static void main (String[] args)

{

int inarr1[] = {1, 2, 3};

int inarr2[] = {1, 2, 3};

Object[] arr1 = {inarr1}; // arr1 contains only one element

Object[] arr2 = {inarr2}; // arr2 also contains only one element

if (Arrays.deepEquals(arr1, arr2))

System.out.println("Same");

else

System.out.println("Not same");

}

}

**12. Write a java code to display a reverse String array.**

import java.lang.\*;

import java.io.\*;

import java.util.\*;

// Class of ReverseString

class ReverseString

{

public static void main(String[] args)

{

String input = "Geeks for Geeks";

StringBuilder input1 = new StringBuilder();

// append a string into StringBuilder input1

input1.append(input);

// reverse StringBuilder input1

input1 = input1.reverse();

// print reversed String

for (int i=0; i<input1.length(); i++)

System.out.print(input1.charAt(i));

}

}

**13.   Write the difference between checked and unchecked exception with example code**

**Ans :-**

**1) Checked:** are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using throws keyword.

For example, consider the following Java program that opens file at locatiobn “C:\test\a.txt” and prints first three lines of it. The program doesn’t compile, because the function main() uses FileReader() and FileReader() throws a checked exception FileNotFoundException. It also uses readLine() and close() methods, and these methods also throw checked exception IOException

import java.io.\*;

class Main {

    public static void main(String[] args) {

        FileReader file = new FileReader("C:\\test\\a.txt");

        BufferedReader fileInput = new BufferedReader(file);

        // Print first 3 lines of file "C:\test\a.txt"

        for (int counter = 0; counter < 3; counter++)

            System.out.println(fileInput.readLine());

        fileInput.close();

    }

}

To fix the above program, we either need to specify list of exceptions using throws, or we need to use try-catch block. We have used throws in the below program. Since FileNotFoundException is a subclass of IOException, we can just specify IOException in the throws list and make the above program compiler-error-free.

**Fix ussue**

import java.io.\*;

class Main {

    public static void main(String[] args) throws IOException {

        FileReader file = new FileReader("C:\\test\\a.txt");

        BufferedReader fileInput = new BufferedReader(file);

        // Print first 3 lines of file "C:\test\a.txt"

        for (int counter = 0; counter < 3; counter++)

            System.out.println(fileInput.readLine());

        fileInput.close();

    }

}

**2) Unchecked** are the exceptions that are not checked at compiled time. In C++, all exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized, and specify or catch the exceptions.  
In Java exceptions under Error and RuntimeException classes are unchecked exceptions, everything else under throwable is checked.

Consider the following Java program. It compiles fine, but it throws ArithmeticExceptionwhen run. The compiler allows it to compile, because ArithmeticException is an unchecked exception.

class Main {

   public static void main(String args[]) {

      int x = 0;

      int y = 10;

      int z = y/x;

  }

}

**14.   Write the difference between throw and throws with example code**

**Ans :-**

1. **Throws clause** is used to declare an exception, which means it works similar to the try-catch block. On the other hand **throw** keyword is used to throw an exception explicitly.

2. If we see syntax wise than **throw** is followed by an instance of Exception class and **throws** is followed by exception class names.  
For example:

throw new ArithmaticException(“Arithmatic Exception……”)

throws Arithmetic Exception

3. Throw keyword is used in the method body to throw an exception, while throws is used in method signature to declare the exceptions that can occur in the statements present in the method.

For example:  
**Throw:**

**Void display()**

**{**

**try{**

**throw new ArithmaticException(“something went wrong…”);**

**}**

**catch(Exception e)**

**{System.out.println(e);**

**}**

**}**

Throws :

Void display()throws ArithmaticException

Class demo throws Arithamatic Exception,IOException{}

**15.   Write a note or nested try…catch block with example code**

**Ans:-**

## Try block

The try block contains set of statements where an exception can occur. A try block is always followed by a catch block, which handles the exception that occurs in associated try block. A try block must be followed by catch blocks or finally block or both.

**try{**

**Statements;**

**}**

While writing a program, if you think that certain statements in a program can throw a exception, enclosed them in try block and handle that exception

## Catch block

A catch block is where you handle the exceptions, this block must follow the try block. A single try block can have several catch blocks associated with it. You can catch different exceptions in different catch blocks. When an exception occurs in try block, the corresponding catch block that handles that particular exception executes. For example if an arithmetic exception occurs in try block then the statements enclosed in catch block for arithmetic exception executes.

Try{

Statements;

}

Catch(Exception obj)

{

System.out.println(obj+msg);

}

Example

**public** **class** Testtrycatch2{

**public** **static** **void** main(String args[]){

**try**{

**int** data=50/0;

   }**catch**(ArithmeticException e){System.out.println(e);}

   System.out.println("rest of the code...");

}

}

**16.   Write a note on MultiThreading and MultiTasking**

**Ans:- Multithreading in java** is a process of executing multiple threads simultaneously.

Thread is basically a lightweight sub-process, a smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

But we use multithreading than multiprocessing because threads share a common memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation etc.

### Advantages of Java Multithreading

### 1) It doesn't block the user because threads are independent and you can perform multiple operations at same time.

2) You **can perform many operations together so it saves time**.

3) Threads are **independent** so it doesn't affect other threads if exception occur in a single thread.

**Multitasking** is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU. Multitasking can be achieved by two ways:

1. Process-based Multitasking(Multiprocessing)
2. Thread-based Multitasking(Multithreading)

**17.   Write a short note on Deque and give example code.**

The java.util.Deque interface is a subtype of the [java.util.Queue](https://www.geeksforgeeks.org/queue-interface-java/) interface. The Deque is related to the double-ended queue that supports addition or removal of elements from either end of the data structure, it can be used as a [queue (first-in-first-out/FIFO)](https://www.geeksforgeeks.org/queue/) or as a [stack (last-in-first-out/LIFO)](https://www.geeksforgeeks.org/stack/).

**Methods** of deque:

1. **add(element):**Adds an element to the tail.
2. **addFirst(element):**Adds an element to the head.
3. **addLast(element):** Adds an element to the tail.
4. **offer(element):**Adds an element to the tail and returns a boolean to explain if the insertion was successful.
5. **offerFirst(element):**Adds an element to the head and returns a boolean to explain if the insertion was successful.
6. **offerLast(element):**Adds an element to the tail and returns a boolean to explain if the insertion was successful.
7. **iterator():** Returna an iterator for this deque.
8. **descendingIterator():** Returns an iterator that has the reverse order for this deque.
9. **push(element):**Adds an element to the head.
10. **pop(element):** Removes an element from the head and returns it.
11. **removeFirst():**Removes the element at the head.
12. **removeLast():** Removes the element at the tail.

|  |
| --- |
| // Java program to demonstrate working of  // Deque in Java  import java.util.\*;    public class DequeExample  {      public static void main(String[] args)      {          Deque deque = new LinkedList<>();            // We can add elements to the queue in various ways          deque.add("Element 1 (Tail)"); // add to tail          deque.addFirst("Element 2 (Head)");          deque.addLast("Element 3 (Tail)");          deque.push("Element 4 (Head)"); //add to head          deque.offer("Element 5 (Tail)");          deque.offerFirst("Element 6 (Head)");          deque.offerLast("Element 7 (Tail)");            System.out.println(deque + "\n");            // Iterate through the queue elements.          System.out.println("Standard Iterator");          Iterator iterator = deque.iterator();          while (iterator.hasNext())              System.out.println("\t" + iterator.next());              // Reverse order iterator          Iterator reverse = deque.descendingIterator();          System.out.println("Reverse Iterator");          while (reverse.hasNext())              System.out.println("\t" + reverse.next());            // Peek returns the head, without deleting          // it from the deque          System.out.println("Peek " + deque.peek());          System.out.println("After peek: " + deque);            // Pop returns the head, and removes it from          // the deque          System.out.println("Pop " + deque.pop());          System.out.println("After pop: " + deque);            // We can check if a specific element exists          // in the deque          System.out.println("Contains element 3: " +                          deque.contains("Element 3 (Tail)"));            // We can remove the first / last element.          deque.removeFirst();          deque.removeLast();          System.out.println("Deque after removing " +                              "first and last: " + deque);        }  } |

**18.   Write a short note on Generics an all types of Parameters used in Generics with example code.**

The **Java Generics** programming is introduced in J2SE 5 to deal with type-safe objects.

Before generics, we can store any type of objects in collection i.e. non-generic. Now generics, forces the java programmer to store specific type of objects.

#### Advantage of Java Generics

There are mainly 3 advantages of generics. They are as follows:

**1) Type-safety :** We can hold only a single type of objects in generics. It doesn’t allow to store other objects.

**2) Type casting is not required:** There is no need to typecast the object.

Before Generics, we need to type cast.

List list = **new** ArrayList();

list.add("hello");

String s = (String) list.get(0);//typecasting

**After genric**

List<String> list = **new** ArrayList<String>();

list.add("hello");

String s = list.get(0);

**Syntax** to use generic collection

ClassOrInterface<Type>

**Example** to use Generics in java

ArrayList<String>

Here, we are using the ArrayList class, but you can use any collection class such as ArrayList, LinkedList, HashSet, TreeSet, HashMap, Comparator etc.

**import** java.util.\*;

**class** TestGenerics1{

**public** **static** **void** main(String args[]){

ArrayList<String> list=**new** ArrayList<String>();

list.add("chetana");

list.add("nishant");

//list.add(32);//compile time error

String s=list.get(1);//type casting is not required

System.out.println("element is: "+s);

Iterator<String> itr=list.iterator();

**while**(itr.hasNext()){

System.out.println(itr.next());

}

}

}

## Generic class

A class that can refer to any type is known as generic class. Here, we are using **T** type parameter to create the generic class of specific type.

Let’s see the simple example to create and use the generic class.

**class** MyGen<T>{

T obj;

**void** add(T obj){**this**.obj=obj;}

T get(){**return** obj;}

}

Example

**class** TestGenerics3{

**public** **static** **void** main(String args[]){

MyGen<Integer> m=**new** MyGen<Integer>();

m.add(2);

//m.add("vivek");//Compile time error

System.out.println(m.get());

}}

## Type Parameters

The type parameters naming conventions are important to learn generics thoroughly. The commonly type parameters are as follows:

1. T - Type
2. E - Element
3. K - Key
4. N - Number
5. V - Value

## Generic Method

Like generic class, we can create generic method that can accept any type of argument.

Let’s see a simple example of java generic method to print array elements. We are using here **E** to denote the element.

**public** **class** TestGenerics4{

**public** **static** < E > **void** printArray(E[] elements) {

**for** ( E element : elements){

            System.out.println(element );

         }

         System.out.println();

    }

**public** **static** **void** main( String args[] ) {

        Integer[] intArray = { 10, 20, 30, 40, 50 };

        Character[] charArray = { ‘N’,’I’,’I’,’T’ };

        System.out.println( "Printing Integer Array" );

        printArray( intArray  );

       System.out.println( "Printing Character Array" );

        printArray( charArray );

    }

}

## Wildcard in Java Generics

The ? (question mark) symbol represents wildcard element. It means any type. If we write <? extends Number>, it means any child class of Number e.g. Integer, Float, double etc. Now we can call the method of Number class through any child class object.

Let's understand it by the example given below:

**import** java.util.\*;

**abstract** **class** Shape{

**abstract** **void** draw();

}

**class** Rectangle **extends** Shape{

**void** draw(){System.out.println("drawing rectangle");}

}

**class** Circle **extends** Shape{

**void** draw(){System.out.println("drawing circle");}

}

**class** GenericTest{

//creating a method that accepts only child class of Shape

**public** **static** **void** drawShapes(List<? **extends** Shape> lists){

**for**(Shape s:lists){

s.draw();//calling method of Shape class by child class instance

}

}

**public** **static** **void** main(String args[]){

List<Rectangle> list1=**new** ArrayList<Rectangle>();

list1.add(**new** Rectangle());

List<Circle> list2=**new** ArrayList<Circle>();

list2.add(**new** Circle());

list2.add(**new** Circle());

drawShapes(list1);

drawShapes(list2);

}}

**19.   Write a short note on Map Interface.**

**Ans:-**

A map contains values on the basis of key i.e. key and value pair. Each key and value pair is known as an entry. Map contains only unique keys.

Map is useful if you have to search, update or delete elements on the basis of key.

Object put(Object key, Object value)

void putAll(Map map)

Object remove(Object key)

Object get(Object key)

boolean containsKey(Object key)

Set keySet()

Set entrySet()

## Map.Entry Interface

Entry is the sub interface of Map. So we will be accessed it by Map.Entry name. It provides methods to get key and value.

### Methods of Map.Entry interface

Methods

Object getKey()

Object getValue()

Example

**import** java.util.\*;

**class** MapInterfaceExample{

**public** **static** **void** main(String args[]){

  Map<Integer,String> map=**new** HashMap<Integer,String>();

  map.put(100,"chetana");

  map.put(101,"nishant");

  map.put(102,"maru");

**for**(Map.Entry m:map.entrySet()){

   System.out.println(m.getKey()+" "+m.getValue());

  }

 }

}

**20.   Write the difference between LinkedList and ArrayList.**

**Ans : -**

**ArrayList:-**

1. ArrayList internally uses **dynamic array** to store the elements.
2. Manipulation is slow because its uses array internally so if elements are removed then it shift bits.
3. Its act as list because implements List interface
4. Better for sorting and accessing data

**LinkedList:-**

1. LinkedList internally uses **doubly linked list** to store the elements.
2. **Faster than ArrayList**
3. **Act as list & queue because implements both**
4. **Better for data manipulation**

**21.   Write a note on Dynamic array in java.**

**Ans:-**

In Java, the size of an array is fixed when it is created. Elements are not allowed to be inserted or removed. However, it is possible to implement a dynamic array by allocating a new array and copying the contents from the old array to the new one.A *dynamic array* has variable size and allows elements to be added or removed. For this, we can allocate a fixed-size array and divide it into two parts:

* the first part stores the elements of the dynamic array and
* the second part is reserved, but not used.

Then we can add or remove elements at the end of the array by using the reserved space, until this space is completely consumed. After that, we create a bigger array and copy the contents of the old array to the new one.

Logical size (size): the number of elements in the dynamic array

Capacity: the physical size of the internal array (the maximum possible size without relocating storage)

We now design a class DynamicArray represents dynamic arrays of integers. It has two attributes:

int[] data: an integer array, and

int size: the logical size, the number of elements used

The capacity of this dynamic array is simply data.length.

An important method we need is to add elements to the end of the dynamic array. This method should provide automatic extension if the capacity is not large enough to hold the added element.

In summary, we wish to design the class DynamicArray with the following members:

Attributes / Constructors / Methods:

int[] data: the array storing the elements

int size: the number of elements

DynamicArray(): initialize this dynamic array with size 0

DynamicArray(int capacity): initialize this dynamic array with the capacity

int get(int index): get the element at the specified index

int set(int index, int element): set the value of the element at the specified index

boolean add(int element): add the element to the end of the array

void ensureCapacity(int minCapacity): increase the capacity

int size(): return the size of the dynamic array

boolean isEmpty(): check whether the array is empty

void clear(): clean up the elements

**22.   What is the purpose of the System class?**

**Ans: -**

the **System class** are standard input, standard output, and error output streams; access to externally defined properties and environment variables; a means of loading files and libraries; and a utility method for quickly copying a portion of an array.

**23.   Which is the abstract parent class of FileWriter ?**

**Ans:- OutputStreamWriter**

**24.   Which class is used to read streams of characters from a file?**

**Ans:- FileReader**

**25.   Which class is used to read streams of raw bytes from a file?**

**Ans:- FileinputStream**

**26.   What are the differences between FileInputStream/FileOutputStream and RandomAccessFile**

**Ans:-**

-The File class encapsulates the files and directories of the local file system.  
  
- The java.io.RandomAccessFile class implements a random access file.  
  
- Random access file offers a seek feature that can go directly to a particular position.  
  
- Unlike the input and output stream classes in java.io, RandomAccessFile is used for both reading and writing files.  
  
- RandomAccessFile does not inherit from InputStream or OutputStream. It implements the DataInput and DataOutput interfaces.

**27.   Write a note on Channels and Buffer with example.**

**Ans:-**

**Buffer** is a block of data that is to be written to a channel or just read from a channel. It is an object that holds data and acts as an endpoint in a NIO channel. Buffer provides a formal mechanism to access data and tracks the read and write processes.

Buffer is one of the main differences between the old Java I/O and the NIO. Previously data is read directly from a stream or written directly into it. Now the data is read from a buffer or written into it. Channels are synonymous to streams in the NIO.

**28.   What is the difference between System.out ,System.err and System.in?**

**Ans:- System.out :- used to print output on console.**

**System.in :- used to read values from console.**

**System.err :- used to print error message generated by system on cnsole screen.**

**29.   What is the purpose of the System class?**

**Ans:-**

the **System class** are standard input, standard output, and error output streams; access to externally defined properties and environment variables; a means of loading files and libraries; and a utility method for quickly copying a portion of an array.

**30.   Which is the abstract parent class of FileWriter ?**

**Ans:- OutputStreamWriter**

**31.   Which class is used to read streams of characters from a file?**

**Ans:- FileReader class**

**32.   Which class is used to read streams of raw bytes from a file?**

**Ans:-FileInputStream.**

**33.   What are the differences between FileInputStream/FileOutputStream and RandomAccessFile**

**Ans:- see question number 26.**

**35.   Write a note on PreparedStatement and ResultSetMetaData interfaces with code snippets.**

**Ans:-**

**36.   Write a note on DDL, DML, DQL, DDL with code snippets.**

**Ans :-**

**All above are statement of structure query language which is used to communicate with rdbms this are also known as intermediate language.**

**DDL :- its data definition used to create table object alter abject drop object. It has mainly**

**Create,alter,drop,truncate command statements.**

**DML :- its data Manipuation language used to insert,update,delete records from table of RDBMS.**

**It has mainly**

**Insert,update,delete,merge command statements.**

**DQL:- it’s Data query language used to fetch records from database table**

**Select command statements used for it.**

**37.   Write a note on HTML , CSS and Javascript.**

**Ans:-**

**HTML:-Hyper text markup language used to create static web page its collection of tags using that we can create web pages.**

**JavaScript :- is used for creating dynamic or responsive web application.**

**CSS: - cascading style sheet used for give graphical look & feel to web pages.**

**38.   Write a code to fetch the data from H2 and put it in any collection object and display it.**

CREATE TABLE  "USER3333"

   (    "ID" NUMBER,

    "NAME" VARCHAR2(4000),

    "PASSWORD" VARCHAR2(4000),

    "EMAIL" VARCHAR2(4000),

     CONSTRAINT "USER3333\_PK" PRIMARY KEY ("ID") ENABLE

   )

User.java

**public** **class** User {

**private** **int** id;

**private** String name,password,email;

//getters and setters

}

**import** java.sql.\*;

**import** java.util.ArrayList;

**public** **class** FetchRecords {

ArrayList<User> list=**new** ArrayList<User>();

**public** ArrayList<User> getList() {

**return** list;

}

**public** **void** setList(ArrayList<User> list) {

**this**.list = list;

}

**public** String execute(){

**try**{

  Connection con=DriverManager.getConnection(“jdbc:h2:tcp://localhot/~/stud”,”sa”,””);

  PreparedStatement ps=con.prepareStatement("select \* from user3333");

  ResultSet rs=ps.executeQuery();

**while**(rs.next()){

   User user=**new** User();

   user.setId(rs.getInt(1));

   user.setName(rs.getString(2));

   user.setPassword(rs.getString(3));

   user.setEmail(rs.getString(4));

   list.add(user);

  }

  con.close();

 }**catch**(Exception e){e.printStackTrace();}

**return** "success";

}

}

**39.   Describe the different approaches of String processing.**

**Ans:-**

**40.   What is the difference between System.out ,System.err and System.in?**

**Ans:- see question number 27**

**41.   What is the purpose of the System class?**

System class is provided with useful fields (static members) pertaining to the environment.Standard input,output and error output streams are provided with System class. These are used to access the externally defined properties and environment variables.

Other useful methods that interact with external system / environment are:  
  
- currentTimeMillis() – returns the current time in milliseconds  
- exit() - terminates currently running JVM  
- gc() - invokes the garbage collector  
- getProperties() - returns the system properties.

**42.   Which is the abstract parent class of FileWriter ?**

**Ans:- repeat**

**43.   Which class is used to read streams of characters from a file?**

**Ans:- repeat**

**44.   Which class is used to read streams of raw bytes from a file?**

**45.   What are the differences between FileInputStream/FileOutputStream and RandomAccessFile**

**Ans:- see question no 26.**

**46.   Write a note on Channels and Buffer with example.**

**Ans:-**NIO allows Java programmers to implement high-speed I/O without having to write custom native code. NIO moves the most time consuming I/O activities (like, filling and draining buffers) back into the operating system, thus providing increase in speed. NIO makes I/O faster than traditional I/O.

Java NIO library implements the NIO APIs. Java NIO is not a substitute to Java IO, rather is a complement to it. Since Java 1.4, there have been enhancements in NIO (and IO) in Java SE 5, 6 and 7 versions.

Buffers and channels are the main concepts of the NIO.

**Buffer:**A buffer is essentially a block of memory into which you can write data, which you can then later read again. This memory block is wrapped in a NIO Buffer object, which provides a set of methods that makes it easier to work with the memory block.

Using a Buffer to read and write data typically follows this little 4-step process:

1. Write data into the Buffer
2. Call buffer.flip()
3. Read data out of the Buffer
4. Call buffer.clear() or buffer.compact()

A Buffer has three properties you need to be familiar with, in order to understand how a Buffer works. These are:

* capacity
* position
* limit

The meaning of position and limit depends on whether the Buffer is in read or write mode. Capacity always means the same, no matter the buffer mode.

Java NIO comes with the following **Buffer** types:

* ByteBuffer
* MappedByteBuffer
* CharBuffer
* DoubleBuffer
* FloatBuffer
* IntBuffer
* LongBuffer
* ShortBuffer

Example:

RandomAccessFile aFile = new RandomAccessFile("data/nio-data.txt", "rw");

FileChannel inChannel = aFile.getChannel();

//create buffer with capacity of 48 bytes

ByteBuffer buf = ByteBuffer.allocate(48);

**int bytesRead = inChannel.read(buf);** //read into buffer.

while (bytesRead != -1) {

**buf.flip();** //make buffer ready for read

while(buf.hasRemaining()){

System.out.print((char) **buf.get()**); // read 1 byte at a time

}

**buf.clear();** //make buffer ready for writing

bytesRead = inChannel.read(buf);

}

aFile.close();

# Channel:- In Java NIO, channels are used for I/O transfers. ****Channel is a like a tube that transports data between a buffer and an entity at other end.**** A channel reads data from an entity and places it in buffer blocks for consumption. Similarly, we should write to buffer blocks and that data will be transported by the channel to the other end.

Channels are the gateway provided by Java NIO to access the native I/O mechanism. We should use buffers to interact with the channels, so the channel is like a bridge between two entities to do the I/O. Buffers are the endpoints provided by channels to send and receive data.

## Channel Characteristics

* Unlike streams, channels are two-way. A channel can both read and write.
* Channel reads data into a buffer and writes data from a buffer.
* Channels can do asynchronous read and write operations.
* Channels can be on blocking or non-blocking modes.
* Non-blocking channel does not put the invoking thread in sleep mode.
* Stream-oriented channels like sockets only can be placed in non-blocking mode.
* Data can be transferred from Channel to Channel if any one of them is a FileChannel.

## Channel Classes

Following are the two major types of Channels classes provided as implementation in Java NIO package.

* FileChannel
  + These are based File read/write channels that cannot be placed on nonblocking mode.
* SocketChannel
  + There are three socket channel types namely, SocketChannel, ServerSocketChannel and DatagramChannel.
  + These are selectable channels that can operate in nonblocking mode.

# Example:

import java.io.IOException;

import java.io.RandomAccessFile;

import java.nio.ByteBuffer;

import java.nio.channels.FileChannel;

public class ChannelExample {

public static void main(String args[]) throws IOException {

RandomAccessFile file = new RandomAccessFile("temp.txt", "r");

FileChannel fileChannel = file.getChannel();

ByteBuffer byteBuffer = ByteBuffer.allocate(512);

while (fileChannel.read(byteBuffer) > 0) {

// flip the buffer to prepare for get operation

byteBuffer.flip();

while (byteBuffer.hasRemaining()) {

System.out.print((char) byteBuffer.get());

}

// clear the buffer ready for next sequence of read

byteBuffer.clear();

}

file.close();

}

}