1.A manufacturing company has classified its executives into four levels for the benefit of certain perks. The levels and corresponding perks are shown below.

Level	Perks	
	Conveyance Allowance	Entertainment Allowance
Manager	1000	500
Project Leader	750	200
Software Engineer	500	100
Consultant	250	-

Write a program that will read an executives job number, level number and basic pay and then compute the net salary.

Procedure:

1. Design a simple class called as "Employee" with the following variables
   Level as String
   Job_number as integer
   Gross, Basic, House_rent, Perks, Net, Incometax as double
   Initialize the allowance as
   CA1=1000,CA2=750,CA3=500,CA4=250,EA1=500,EA2=200,EA3=100,EA4=0
2. Get job number, Level and Basic pay from user (hint: use scanner class to get input from user)
3. Calculate the perks for different levels (hint: use switch case to select between levels)
4. Calculate gross, House_rent, Incometax and Net as per Specification given.
5. Display Level, Job_number, Gross pay, Tax and Net salary.

Specification:

HRA at 25% of basic pay
Perks = CA1+EA1 (for Manager) etc.,
Gross salary = basic pay+ HRA + Perks
Net salary = Gross Salary – Income Tax
To calculate Income Tax use the following information

Gross salary	Tax Rate
Gross <=2000	No Tax
2000 < Gross <=4000	3%
4000 < Gross <= 5000	5%
Gross> 5000	8%

```
Import java.utilscanner;
Public class employee
{
Scanner sc=new Scanner (system.in);
System.out.println("enter job no:");
```

```
Int j-no=sc.nextint();
System.out.println”Job No:”, j-no);
SYSTEM.OUT.PRINTLN(“enter level:”);
Int level = sc.nextint();
SYSTEM.OUT.PRINTLN(“Level:”,level);
SYSTEM.OUT.PRINTLN( “enter basic pay”);
Int b_pay=sc.nextint();
SYSTEM.OUT.PRINTLN(“basicpay:”b_pay);
h_rent=b_pay*0.25;
switch (level)
{
Case 1 : perks>1500;
Break;
Case 2 : perks>950;
Break;
Case 3 : perks>600;
Break;
Case 4 : perks>250;
Break;
Case 5 : perks>0;
Break;
g_salary=b_pay+h_rent_perks
if (g.salary <=2000)
{
tax=0.0;
}
Else if (g_salary>2000 && g_salary<=4000)
{
Tax= g_salary*0.03;
}
Else if (g_salary>4000 && g_salary<=5000)
{
Tax= g_salary*0.05;
}
Else
{
Tax= g_salary*0.08;
}
Net.salary=g.salary_tax;
Public static void main(string args[])
{
employee emp= new employee();
SYSTEM.OUT.PRINTLN(“perks:”,emp.perks);
SYSTEM.OUT.PRINTLN(“Gross Salary:”,emp.g.salary);
SYSTEM.OUT.PRINTLN(“incometax:”,tax);
SYSTEM.OUT.PRINTLN(“Net salary:”,emp.netsalary);
}
}
```

2.Think of an application which is capable of demonstrating the use of almost all data types. Do write a java program.

```
public class JavaDataTypes {
    public static void main(String[] args) {
        // integer (whole number)
        int a = 10;
        // String
        String b = "Hello";
        // Float
        float c = 10.5f;
        // Double
        double d = 20.56;
        // Boolean
        boolean e = true;
        // Character
        char f = 'a';

        System.out.println(a);
        System.out.println(b);
        System.out.println(c);
        System.out.println(d);
        System.out.println(e);
        System.out.println(f);

        }
        }
```

3.Differentiate between implicit and explicit conversion.

Implicit Type Conversion	Explicit Type Conversion
An implicit type conversion is automatically performed by the compiler when differing data types are intermixed in an expression.	An explicit type conversion is user-defined conversion that forces an expression to be of specific type.

Implicit Type Conversion	Explicit Type Conversion
An implicit type conversion is performed without programmer's intervention.	An explicit type conversion is specified explicitly by the programmer.
Example: a, b = 5, 25.5 c = a + b	Example: a, b = 5, 25.5 c = int(a + b)

4.Declare and initialize the array variable using new keyword.

You can declare the array with the syntax below:

dataType [ ] nameOfArray;
dataType: the type of data you want to put in the array. This could be a string, integer, double, and so on.
[ ]: signifies that the variable to declare will contain an array of values
nameOfArrary: The array identifier.
With the above information, you have only declared the array – you still need to initialize it.

The basic syntax for initializing an array in this way looks like this:

dataType [] nameOfArray = new dataType [size]
The size is usually expressed with a numberic value. It signifies how many values you want to hold in the array. Its value is immutable, meaning you won't be able to put more than the number specified as the size in the array.

You can now go ahead and put values in the array like this:

```
package com.kolade;

import java.util.Arrays;

public class Main {

    public static void main(String[] args) {
   // write your code here
```

```
String [] names = new String[3];
names[0] = "Quincy";
names[1] = "Abbey";
names[2] = "Kolade";
  }
}
```

In the code snippet above, I initialized an array of strings called names (the identifier). The size is 3, so it can only hold three values.

There are 3 indexes in total:

- The value, Quincy is at index 0
- The value Abbey is at index 1
- The valueKolade is at index 2

Don't be confused by the numbers 0, 1, 2. Arrays are zero-indexed, so counting starts from 0, not 1.

In the array above, if you add extra data – for example, names[3] = "Chris" – you would get an error because you have specified that the array should only contain 3 values. If you want to add more values, you have to increase the size of the array.

5.What is the effect of converting an integer to byte? Justify.

When an integer value is converted into a byte, Java cuts-off the left-most 24 bits.

We will be using bitwise AND to mask all of the extraneous sign bits.

Here is an illustration of a Java Program that converts an integer to byte.

```
import java.util.Scanner;
public class Integer_Conversion
{
  public static void main(String[] args)
  {
     int a;
     byte b;
     Scanner s = new Scanner(System.in);
     System.out.print("Enter any integer:");
     a = s.nextInt();
     b = (byte) a;
```

```
        System.out.println("Conversion into byte:"+b);
    }
}
```

Output:

Enter any integer: 100

6.How negative numbers are represented in Java? Justify with an example

Java and most other languages store negative integral numbers in a representation called 2's complement notation.

For a unique binary representation of a data type using n bits, values are encoded like this:

The least significant n-1 bits store a positive integral number x in integral representation. Most significant value

stores a bit vith value s. The value repesented by those bits is

x - s * 2n-1

i.e. if the most significant bit is 1, then a value that is just by 1 larger than the number you could represent with the

other bits (2n-2 + 2n-3 + ... + 21 + 20 = 2n-1 - 1) is subtracted allowing a unique binary representation for each value from - 2n-1 (s = 1; x = 0) to 2n-1 - 1 (s = 0; x = 2n-1 - 1).

This also has the nice side effect, that you can add the binary representations as if they were positive binary numbers:

v1 = x1 - s1 * 2n-1 v2 = x2 - s2 * 2n-1

s1 s2 x1 + x2 overflow addition result

0 0 No x1 + x2 = v1 + v2

0 0 Yes too large to be represented with data type (overflow)

0 1 No x1 + x2 - 2n-1 = x1 + x2 - s2 * 2n-1

= v1 + v2

0 1 Yes (x1 + x2) mod 2n-1 = x1 + x2 - 2n-1

= v1 + v2

1 0 * see above (swap summands)

1 1 No too small to be represented with data type (x1 + x2 - 2n < -2n-1 ; underflow)

1 1 Yes

(x1 + x2) mod 2n-1 - 2n-1 = (x1 + x2 - 2n-1) - 2n-1

= (x1 - s1 * 2n-1) + (x2 - s2 * 2n-1)

= v1 + v2

Note that this fact makes finding binary representation of the additive inverse (i.e. the negative value) easy:

Observe that adding the bitwise complement to the number results in all bits being 1. Now add 1 to make value

overflow and you get the neutral element 0 (all bits 0).

So the negative value of a number i can be calculated using (ignoring possible promotion to int here)

(~i) + 1

Example: taking the negative value of 0 (byte):

The result of negating 0, is 11111111. Adding 1 gives a value of 100000000 (9 bits). Because a byte can only store 8

bits, the leftmost value is truncated, and the result is 00000000

Original Process Result

0 (00000000) Negate -0 (11111111)

11111111 Add 1 to binary 100000000

100000000 Truncate to 8 bits 00000000 (-0 equals 0)