**ResultSetMetaData(interface):**

ResultSetMetaData is an interface in java.sql package of JDBC API which is used to get the metadata about a ResultSet object. Whenever you query the database using SELECT statement, the result will be stored in a ResultSet object. This object will have all the meta data about a ResultSet object like schema name, table name, number of columns, column name, datatype of a column etc.

The **getMetaData()** method of ResultSet interface.

**Example:-**

ResultSet rs= ps.exeuteQuery();

**ResultSetMetaData rsmd=rs.getMetaData();**

System.out.println("Total columns: "+rsmd.getColumnCount());

("Column Name of 1st column: "+rsmd.getColumnName(1));

("Column Type Name of 1st column: "+rsmd.getColumnTypename(1));

  con.close();

}

**2. DatabaseMetaData:**

DatabaseMetaData interface provides methods to get meta data of a database such as name of total number of tables, name of total number of views, database product name, database product version, driver name, etc.

Example:

Coonection con=DriverManager.getConnection();

{ **DatabaseMetaData dbmd=con.getMetaData()**;

System.out.println("Driver Name: "+dbmd.getDriverName());

System.out.println("Driver Version: "+dbmd.getDriverVersion());

System.out.println("UserName: "+dbmd.getUserName());

System.out.println("Database Product Name: "+dbmd.getDatabaseProductName();

System.out.println("Database Product Version: "+dbmd.getDatabaseProductVersion());

con.close();

}

**Integrity** **Constraint**:-

It is set of rule…maintain the quality of information.

**String Class:-**

Example:-

String s1 = ”Akash”;

String s2 = ”Thakare”;

String s3=s1.UpperCase();//String is immutable, so require another class to store change

int i=s1.indexOf(a);

int i=s1.indexOf(‘h’, 3);// Start searching from index 3.

int i=s1.indexOf(“ka”,4)//-||-

int i=s1.lastIndexOf(‘a’);// start from last to serach.

**Special Method**

1-> s1.equals(s2)

2-> s1.equalsIgnoreCase(s2)

3-> s1.compareTo(s2)

Constructors in Java

**Note:** It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class.

A Constructor must have **no** explicit return type.

The **parameterized constructor** is used to provide different values to distinct objects. However, you can provide the same values also

When a object is created and **constructor** is **not** declared, the compiler itself provides a **constructor**. This is known as **default constructor**. It does **not** take any parameter and its aim is to give **default** values i.e zero/null to class or Instance variables.

The **default constructor** is used to provide the default values to the object like 0, null, etc., depending on the type. see example:

1. **class** Student3{
2. **int** id;
3. String name;
4. //method to display the value of id and name
5. **void** display()
6. {System.out.println(id+" "+name);}
7. **public** **static** **void** main(String args[]){
8. //creating objects
9. Student3 s1=**new** Student3();
10. Student3 s2=**new** Student3();
11. //displaying values of the object
12. s1.display();
13. s2.display();
14. }
15. }

Output:-

0 null

0 null

**Wrapper Class:**

class MyClass

{

public static void main(String args[])

{

int i=100;

Integer i1=**Integer.valueOf(i)**;

System.out.println(i1);

int j=**i1.intValue();**

System.out.println(j);

String s="123";

int k= **Integer.parseInt(s);**

System.out.println(k);

Integer i3=Integer.valueOf(s);

System.out.println(i3);s

String s=”Akash”;

Int i=Integer.parseInt(s);// NumberFormatException

String s="123";

float k= Float.parseFloat(s);// Print :-123.0

}

}

Primary Key can't accept null values.

Unique key can accept only one null value.

We can have only one Primary key in a table.

We can have more than one unique key in a table.

**If the array is Local variable ( declared inside a function) ,**

**then it is unknown as it will take garbage value.**

**If it is declared Global or static, then it will be initialized to 0 by default , But in java it is Zero by default.**

// Java pre-increment operation

class A

{public static void main(String... ar){

char ch= 'a';

byte b=10;

short sh= 100;

int i=10;

long l=99999;

float f=20.5f;

double d= 15.5;

System.out.println("Original character " + ch);

System.out.println("Incremented character " + ++ch);

System.out.println("Original byte " + b);

System.out.println("Incremented byte " + ++b);

System.out.println("Original short " + sh);

System.out.println("Incremented short " + ++sh);

System.out.println("Original int " + i);

System.out.println("Incremented int " + ++i);

System.out.println("Original long " + l);

System.out.println("Incremented long " + ++l);

System.out.println("Original float " + f);

System.out.println("Incremented float " + ++f);

System.out.println("Original double " + d);

System.out.println("Incremented double " + ++d);

}

}

Output:

**Original character a**

**Incremented character b**

**Original byte 10**

**Incremented byte 11//byte also increamented by 1**

**Original short 100**

**Incremented short 101**

**Original int 10**

**Incremented int 11**

**Original long 99999**

**Incremented long 100000**

**Original float 20.5**

**Incremented float 21.5//float also increamented by 1**

**Original double 15.5**

**Incremented double 16.5**