Software Design and Analysis
CS-3004
Lecture#07

Dr. Javaria Imtiaz,
Mr. Basharat Hussain,
Mr. Majid Hussain

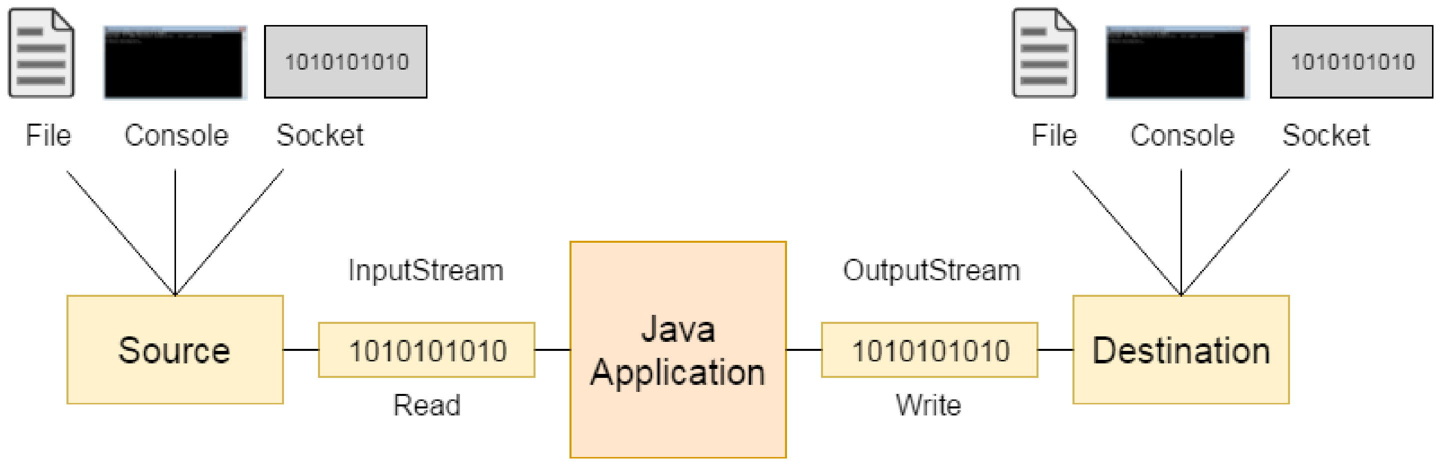
Agenda

- I/O programming
- Streams
- Byte stream
- Character stream
- Data stream

Java I/O

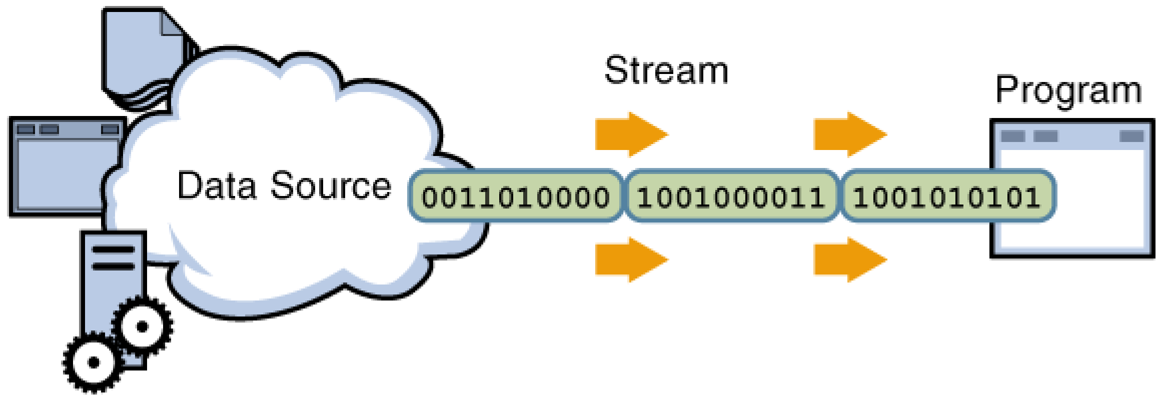
- Java I/O (Input and Output) is used to process the input and produce the output.

- Java uses the concept of a stream to make I/O operation fast. The **java.io** package contains all the classes required for input and output operations.



Stream

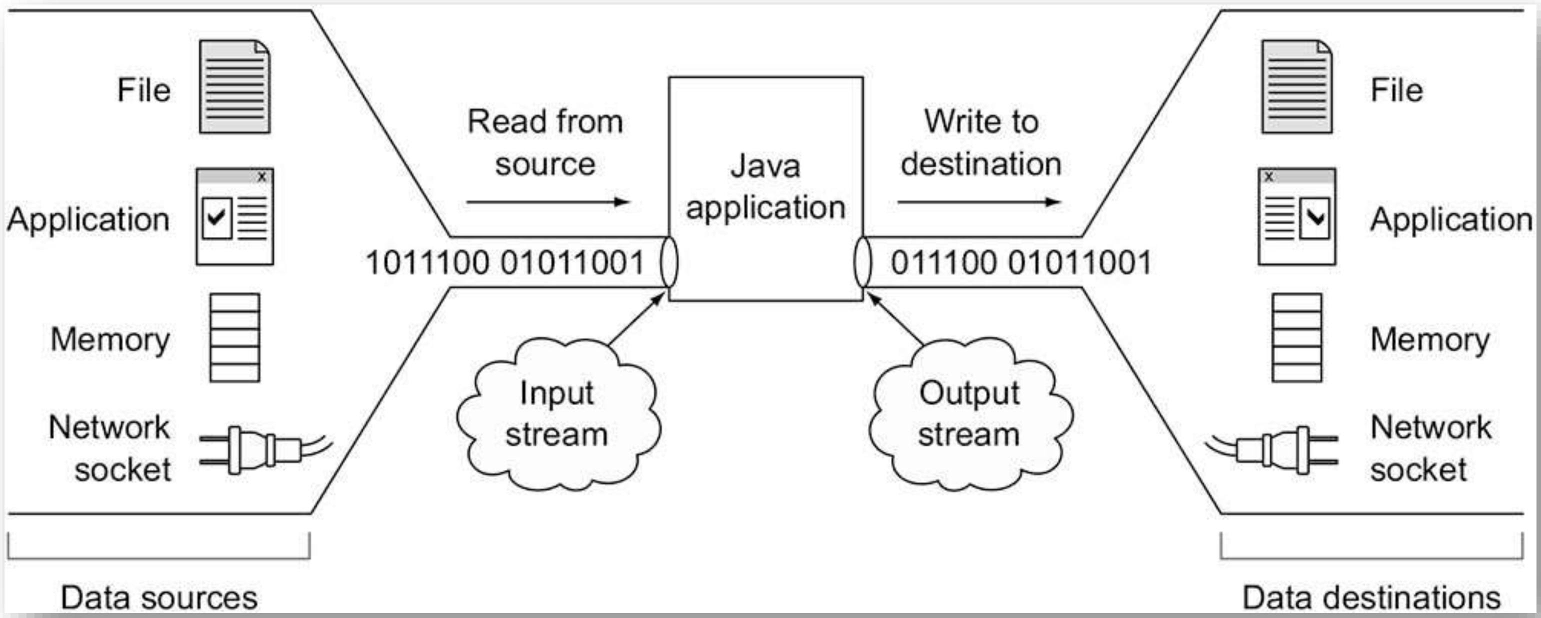
- A stream is a sequence of data. In Java, a stream is composed of bytes. It's called a stream because it is like a stream of water that continues to flow.



Streams

- Streams represents a **Source** (which generates the data in the form of Stream) and a **destination** (which consumes or read data available as Stream).

- Streams supports a huge range of source and destinations including disk file, arrays, other devices, other programs etc.



File Handling

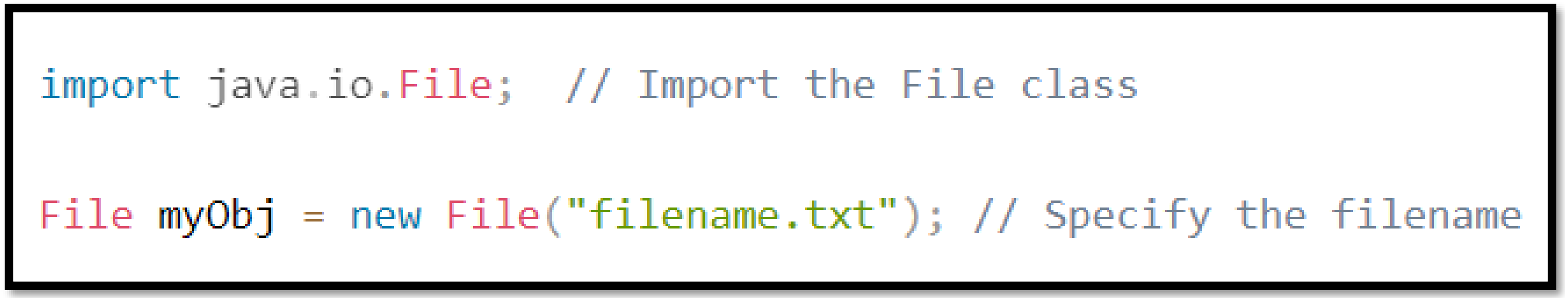
- File handling is an important part of any application.
- File Handling implies how to read from and write to file in Java.
- Java provides the basic I/O package for reading and writing streams.
- **Java.io** package allows to do all Input and Output tasks in Java.
- It provides several methods for creating, reading, updating, and deleting files.

Java’s File Management

- Java class **java.io.File** defines platform-independent manipulation of file system (Files & directories) by providing whether the file
  - exists
  - is read protected
  - is write protected
  - is, in fact, a directory

File Class

- The File class from the java.io package, allows us to work with files.
- To use the File class, create an object of the class, and specify the filename or directory name:



```
import java.io.File;  // Import the File class

File myObj = new File("filename.txt"); // Specify the filename
```

File Class

- A File object can refer to either a file or directory

```
File file1 = new File(“data.txt”);
File file2 = new File(“C:\Java”);
```

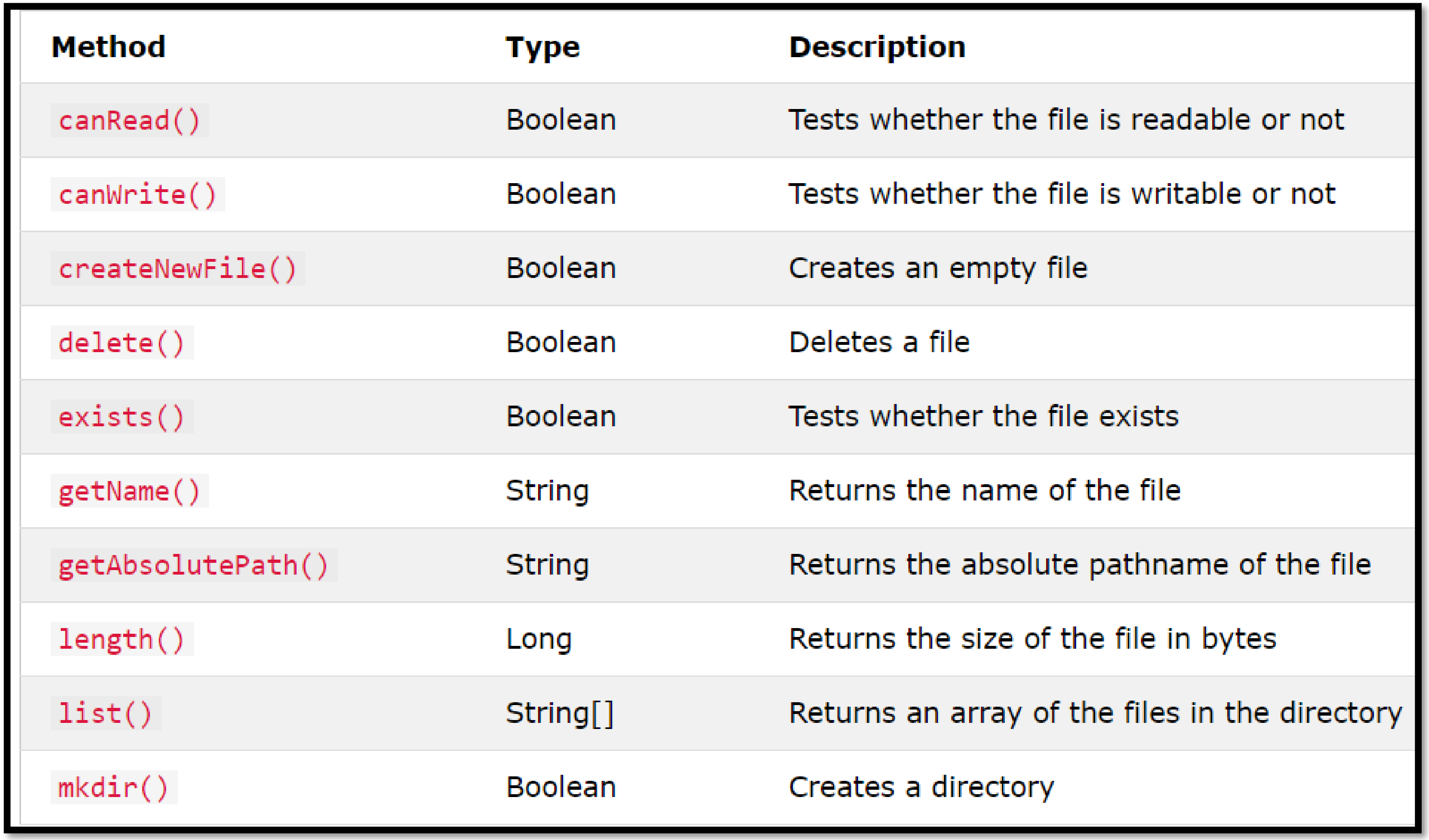
- To obtain the path to the current working directory use

```
System.getProperty(“user.dir”);
```

- To obtain the file or path separator use

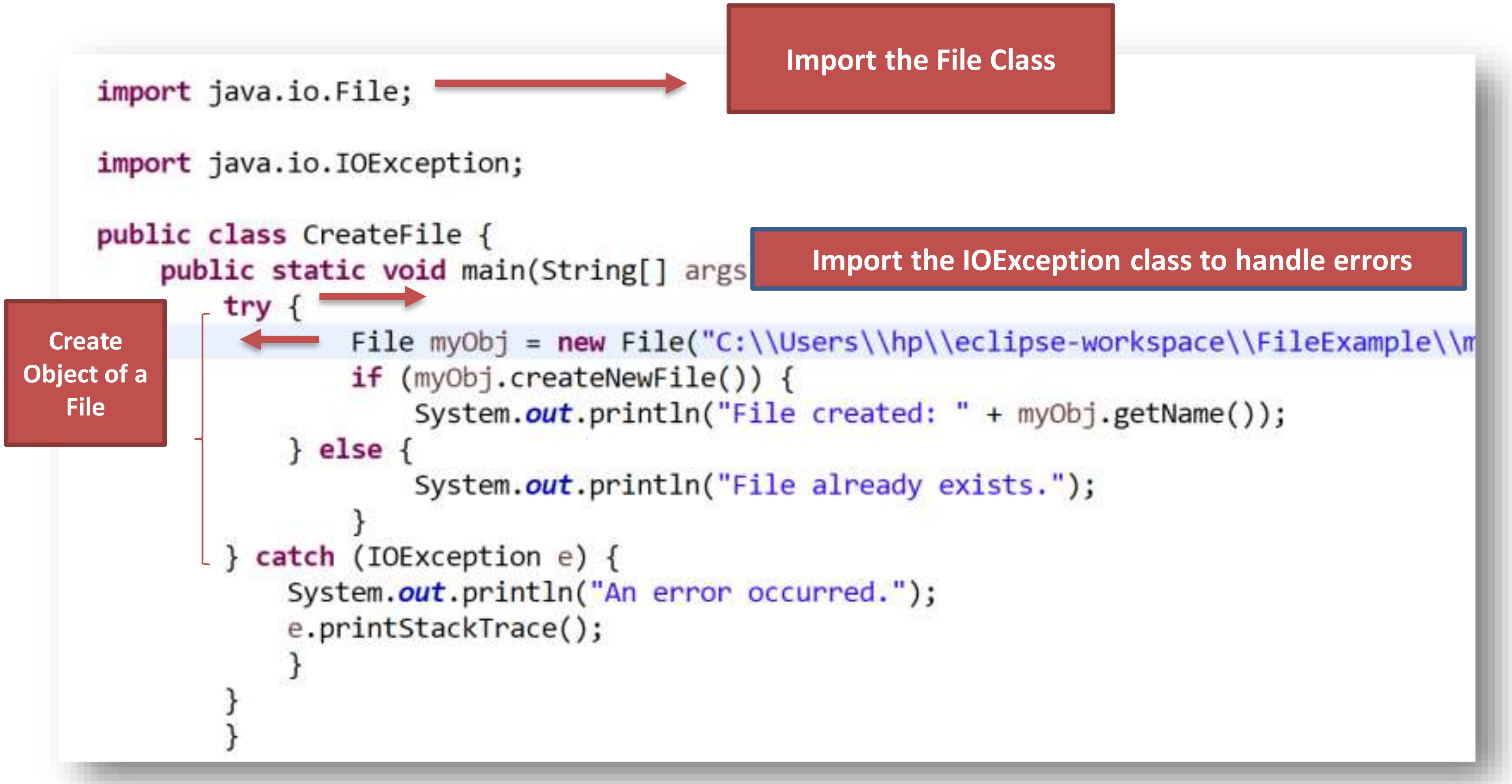
```
System.getProperty (“file.separator”);
System.getProperty (“path.separator”);
```

Useful File Methods



Method	Type	Description
`canRead()`	Boolean	Tests whether the file is readable or not
`canWrite()`	Boolean	Tests whether the file is writable or not
`createNewFile()`	Boolean	Creates an empty file
`delete()`	Boolean	Deletes a file
`exists()`	Boolean	Tests whether the file exists
`getName()`	String	Returns the name of the file
`getAbsolutePath()`	String	Returns the absolute pathname of the file
`length()`	Long	Returns the size of the file in bytes
`list()`	String[]	Returns an array of the files in the directory
`mkdir()`	Boolean	Creates a directory

Create a File

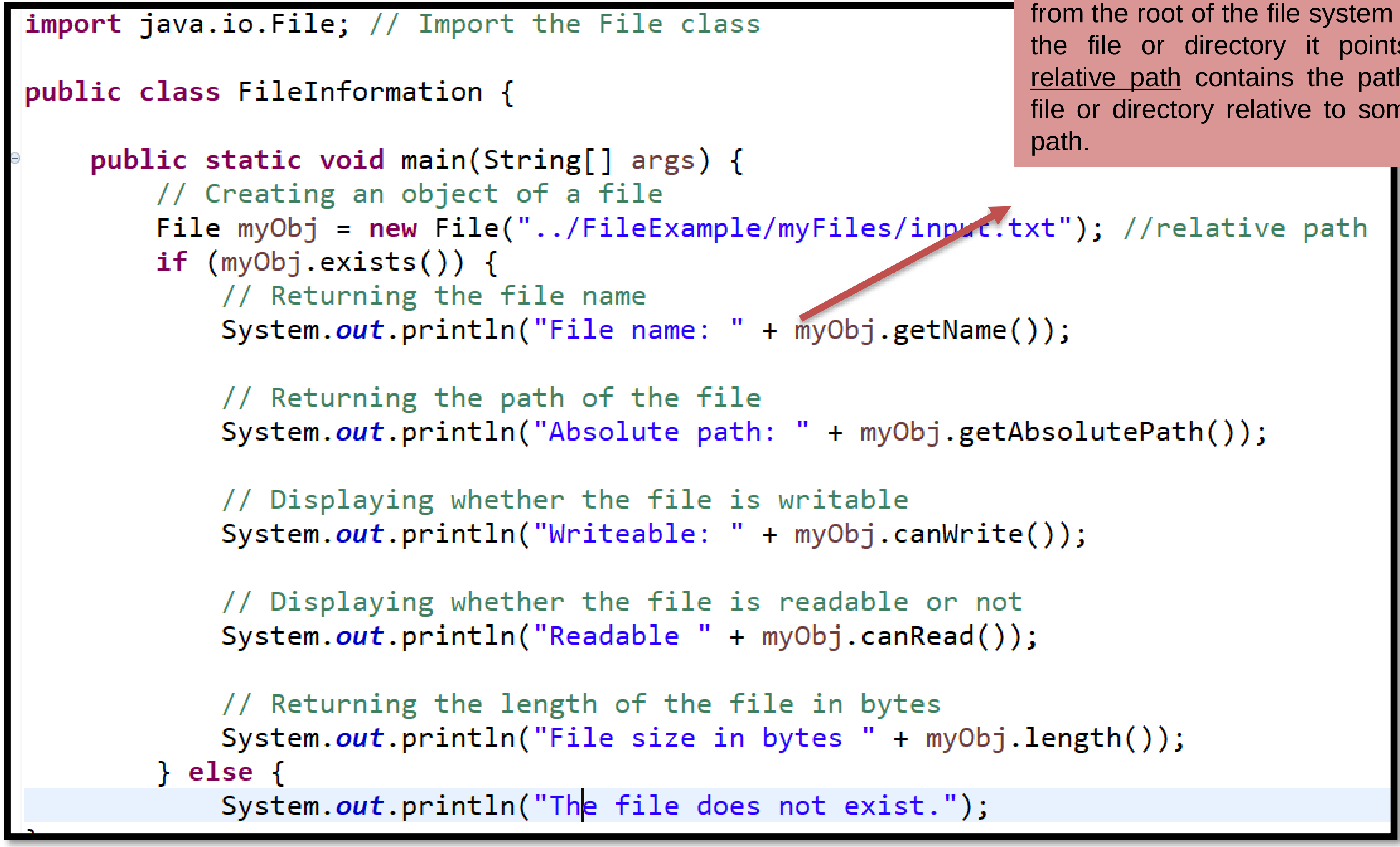


```
import java.io.File;

import java.io.IOException;

public class CreateFile {
    public static void main(String[] args
        try {
            File myObj = new File("C:\\Users\\hp\\eclipse-workspace\\FileExample\\m
            if (myObj.createNewFile()) {
                System.out.println("File created: " + myObj.getName());
        } else {
                System.out.println("File already exists.");
            }
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
            }
        }
    }
```

In **try block**, write a code that has to be executed and **catch block** will handle the errors occur in try block. In this case, the most expected error is IOExpection and that will be handled by the catch block.

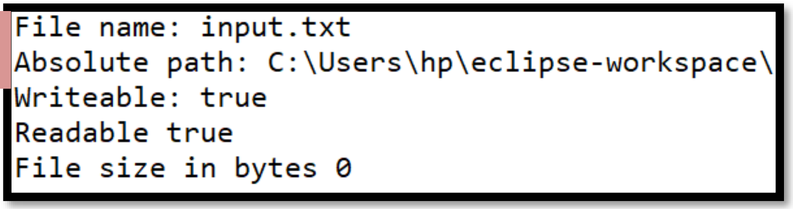
Get File Information

A path can be **absolute** or **relative**. An absolute path contains the full path from the root of the file system down to the file or directory it points to. A relative path contains the path to the file or directory relative to some other path.



```java
import java.io.File; // Import the File class

public class FileInformation {

    public static void main(String[] args) {
        // Creating an object of a file
        File myObj = new File("../FileExample/myFiles/input.txt"); //relative path
        if (myObj.exists()) {
            // Returning the file name
            System.out.println("File name: " + myObj.getName());

            // Returning the path of the file
            System.out.println("Absolute path: " + myObj.getAbsolutePath());

            // Displaying whether the file is writable
            System.out.println("Writeable: " + myObj.canWrite());

            // Displaying whether the file is readable or not
            System.out.println("Readable " + myObj.canRead());

            // Returning the length of the file in bytes
            System.out.println("File size in bytes " + myObj.length());
        } else {
            System.out.println("The file does not exist.");
```

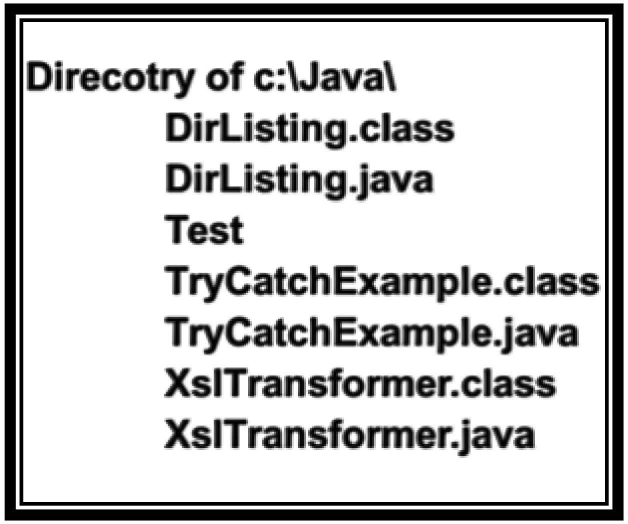
Output



```
File name: input.txt
Absolute path: C:\Users\hp\eclipse-workspace\
Writeable: true
Readable true
File size in bytes 0
```

Directory Listing Example

```
import java.io.*;

public class DirListing {
  public static void main(String[] args) {
    File dir = new File(System.getProperty(“user.dir”));

    if (dir.isDirectory())
        {
      System.out.println(“Directory of ” + dir);
      String[] listing = dir.list();
      for (int i=0; i < listing.length; i++) {
        System.out.println(“\t” + listing[i]);
    }

    }
  }
}
```



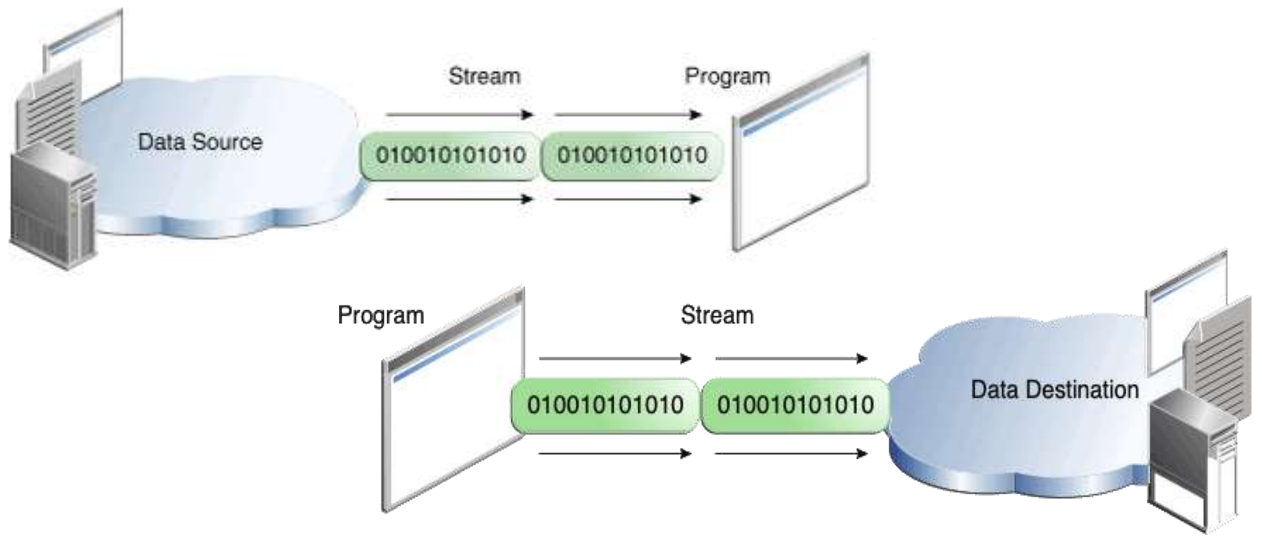
```
Direcotry of c:\Java\
            DirListing.class
            DirListing.java
            Test
            TryCatchExample.class
            TryCatchExample.java
            XslTransformer.class
            XslTransformer.java
```

Directory Listing, Result

```
> java DirListing

Direcotry of c:\Java\
        DirListing.class
        DirListing.java
        Test
        TryCatchExample.class
        TryCatchExample.java
        XslTransformer.class
        XslTransformer.java
```

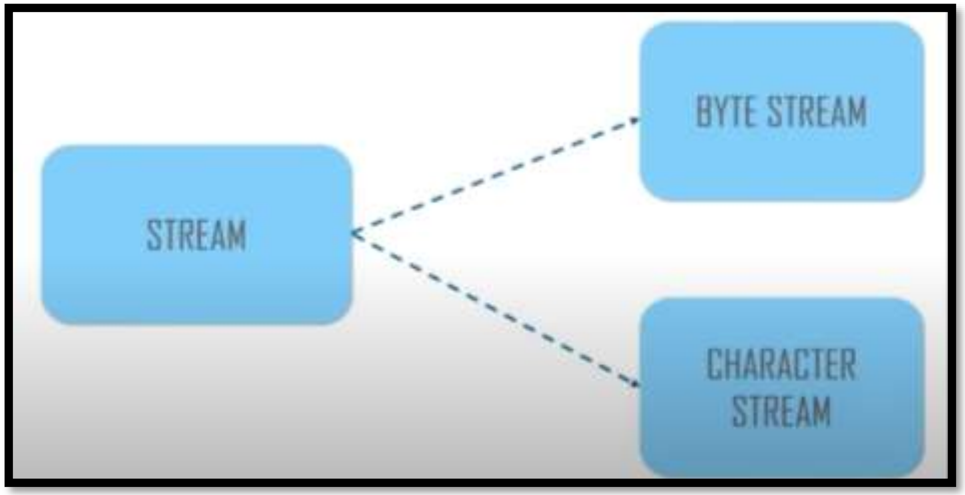
Input/Output Streams

- Java IO package provides over 60 input/output classes (streams)
- Streams are ordered sequence of data that have a source (input), or a destination (output)



Streams

- Java uses the concept of a stream to make I/O operations on a file.



- These handle data in bytes (8 bits) i.e., the byte stream classes read/write data of 8 bits. Using these you can **store characters, videos, audios, images** etc.

- These handle data in 16 bit Unicode. Using these you **can read and write text data** only.

Basic IO Algorithm

- **Reading**

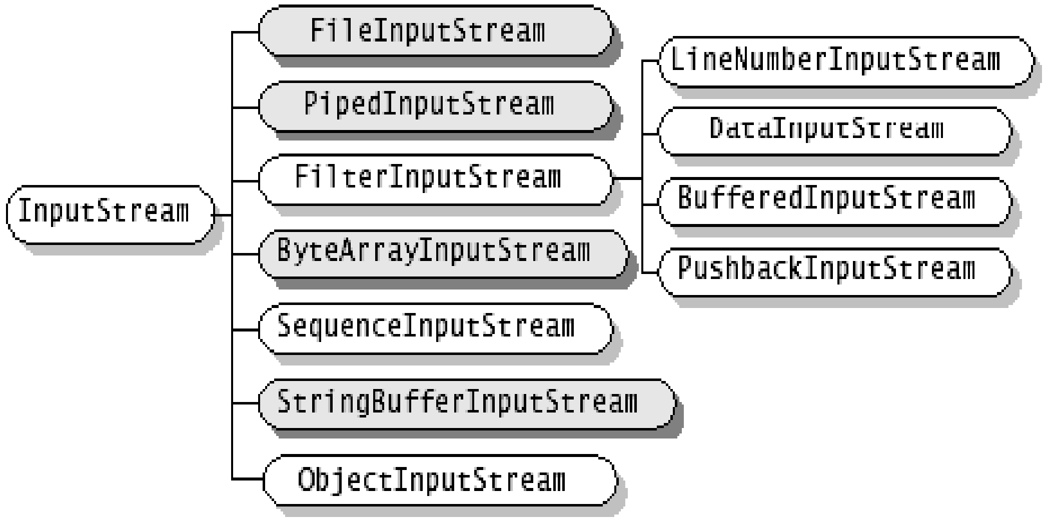
```
open a stream
while more information
    read information

close the stream
```

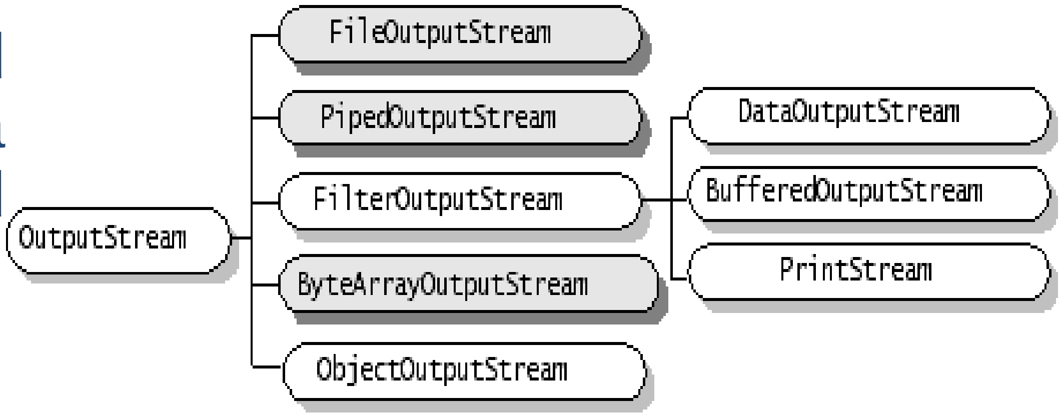
- **Writing**

```
open a stream
while more information
    write information

close the stream
```

Byte Streams

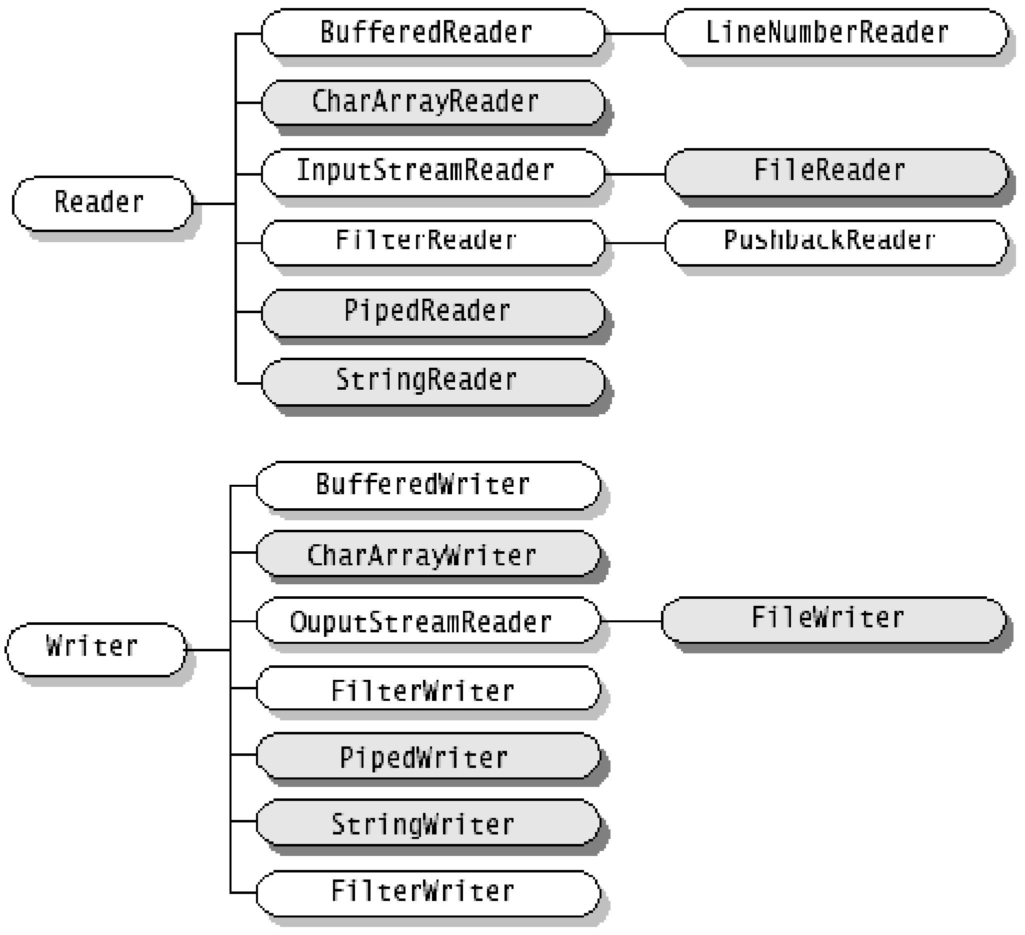
- Read and write 8-bit bytes
- **InputStream** and **OutputStream** are abstract super classes
- Typically used to read and write binary data such as images and sounds





Character Streams

- Read and write 16-bit characters
- **`Reader`** and **`Writer`** are the abstract classes
- Use readers and writers to read and write textual information



I/O Super Classes

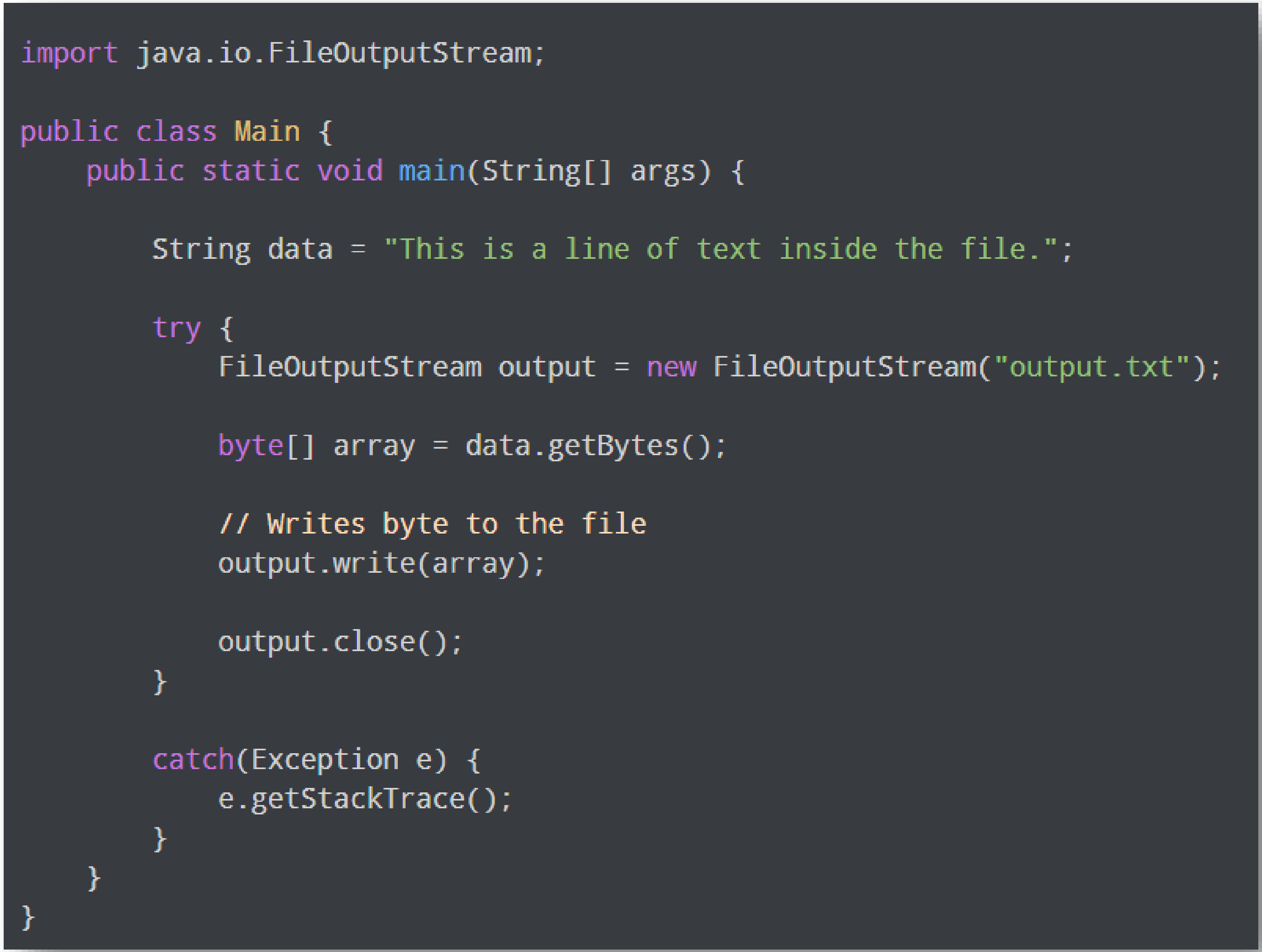
```
Reader
int read();
int read(char cbuf[]);
int read(char cbuf[],
         int offset,
         int length)
```

```
InputStream
int read();
int read(byte buffer[]);
int read(byte buffer[],
         int offset,
         int length)
```

```
Writer
int write(int c);
int write(char cbuf[]);
int write(char cbuf[],
          int offset,
       int length)
```

```
OutputStream
int write(byte b);
int write(byte buffer[]);
int write(byte buffer[],
          int offset,
          int length)
```

FileOutputStream Example



```java
import java.io.FileOutputStream;

public class Main {
    public static void main(String[] args) {

        String data = "This is a line of text inside the file.";

        try {
            FileOutputStream output = new FileOutputStream("output.txt");

            byte[] array = data.getBytes();

            // Writes byte to the file
            output.write(array);

            output.close();
        }

        catch(Exception e) {
            e.getStackTrace();
        }
    }
}
```

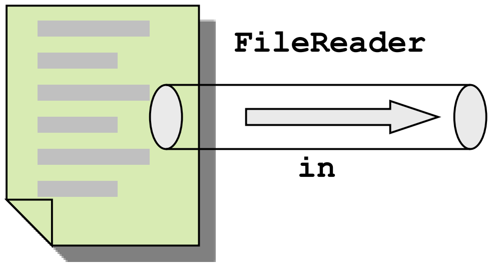
FileInputStream Example

```
This is a line of text inside the file.
```

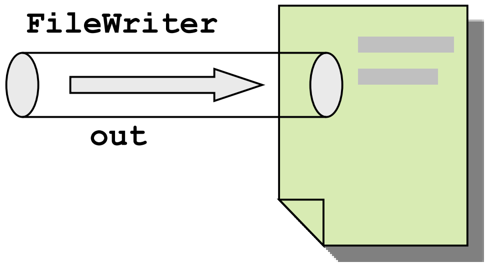
```
import java.io.FileInputStream;

public class Main {

  public static void main(String args[]) {

     try {
        FileInputStream input = new FileInputStream("input.txt");

        System.out.println("Data in the file: ");

        // Reads the first byte
        int i = input.read();

       while(i != -1) {
           System.out.print((char)i);

           // Reads next byte from the file
           i = input.read();
        }
        input.close();
     }

     catch(Exception e) {
        e.getStackTrace();
     }
  }
}
```

Character File Streams

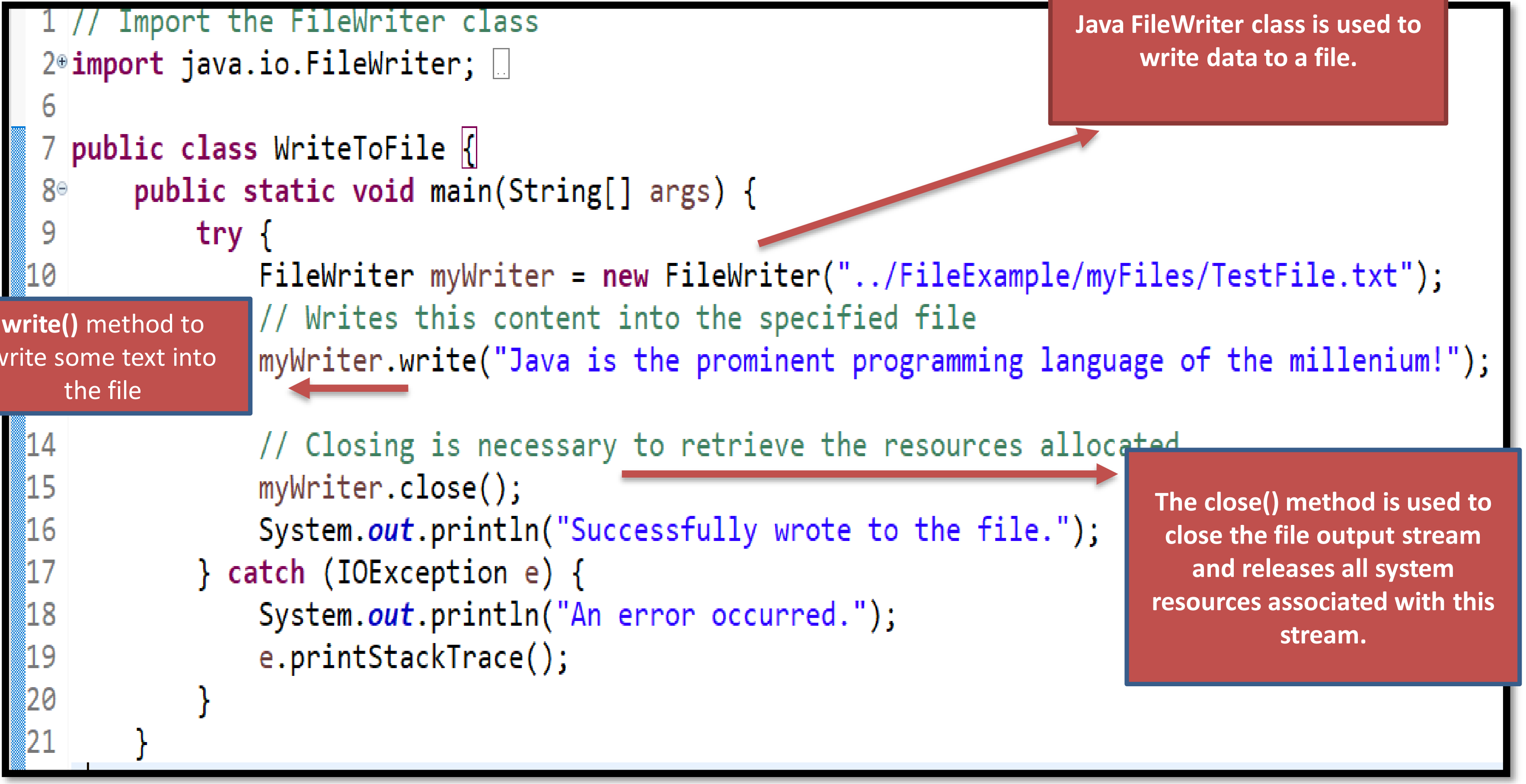
- FileReader



- FileWriter



Write to a file



```java
// Import the FileWriter class
import java.io.FileWriter; 

public class WriteToFile {
    public static void main(String[] args) {
        try {
            FileWriter myWriter = new FileWriter("../FileExample/myFiles/TestFile.txt");
            // Writes this content into the specified file
            myWriter.write("Java is the prominent programming language of the millenium!");

            // Closing is necessary to retrieve the resources allocated
            myWriter.close();
            System.out.println("Successfully wrote to the file.");
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
```

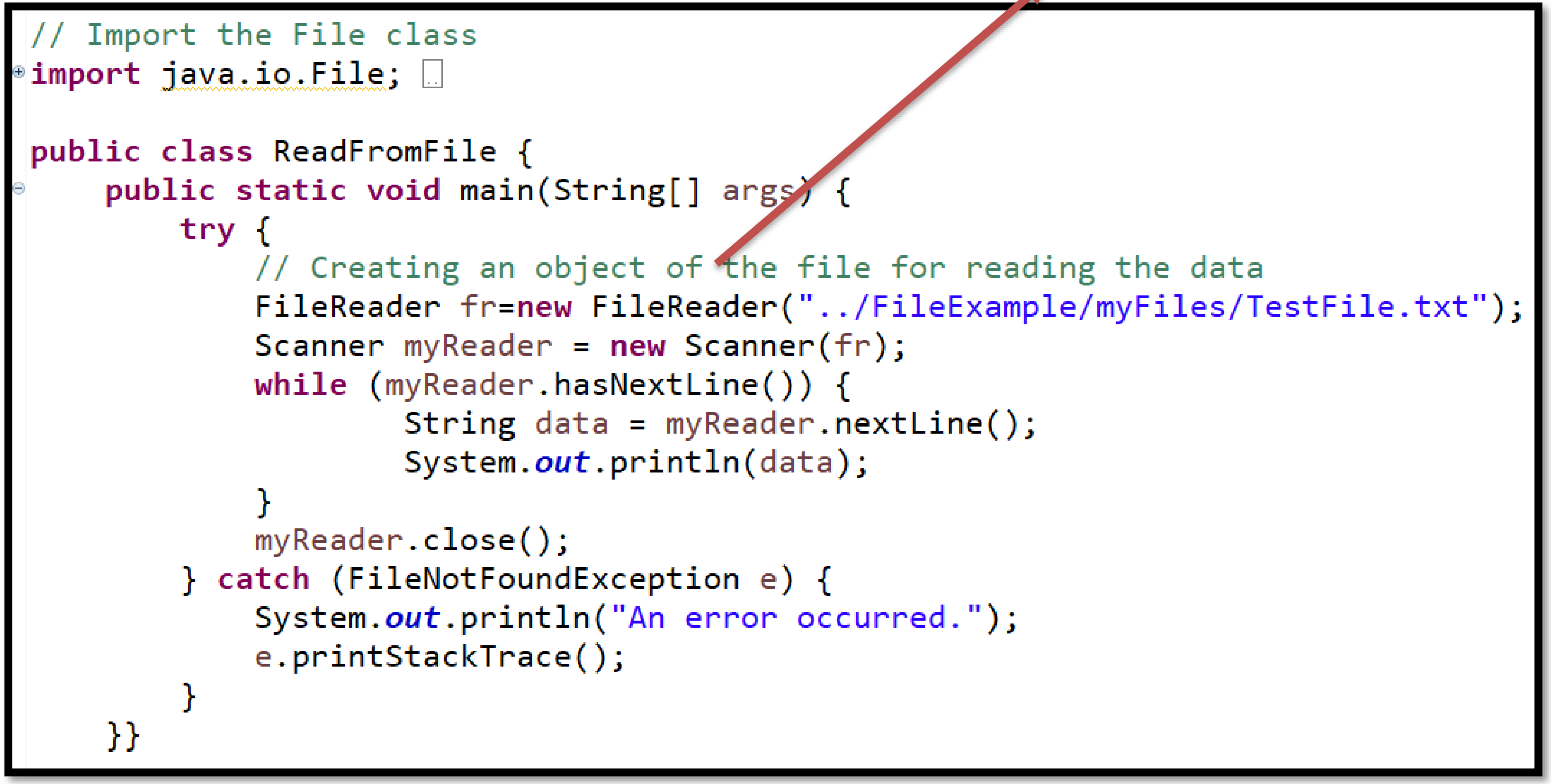
Java FileWriter class is used to write data to a file.

write() method to write some text into the file

The close() method is used to close the file output stream and releases all system resources associated with this stream.

Read a File

FileReader is used for reading streams of characters.



```
// Import the File class
import java.io.File;

public class ReadFromFile {
    public static void main(String[] args) {
        try {
            // Creating an object of the file for reading the data
            FileReader fr=new FileReader("../FileExample/myFiles/TestFile.txt");
            Scanner myReader = new Scanner(fr);
            while (myReader.hasNextLine()) {
                    String data = myReader.nextLine();
                    System.out.println(data);
            }
            myReader.close();
        } catch (FileNotFoundException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }}
```

Buffer Streams

- Buffer data while reading or writing, thereby reducing the number of accesses required on the original data source.
- More efficient than similar non-buffered streams and are often used with other streams
- The buffer size may be specified, or default size may be accepted

Character Streams

```
BufferedReader
BufferedWriter
```

Byte Streams

```
BufferedInputStream
BufferedOutputStream
```

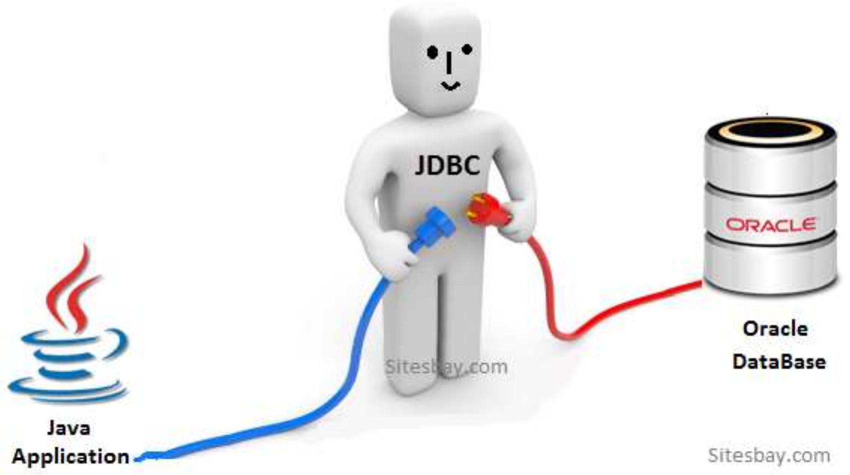
Using Buffer Streams

```
import java.io.*;

public class Copy {
    public static void main(String[] args) throws IOException {
        // opening the streams
        FileReader in = new FileReader (“infile.txt");
        BufferedReader br = new BufferedReader(in);

        FileWriter out = new FileWriter ("outfile.txt");
        BufferedWriter bw = new BufferedWriter(out);

        // processing the streams
        String aLine = null;
        while ((aLine = br.readLine()) != null) {
            bw.write(aLine, 0, aLine.length());
        }
        // closing the streams
        in.close();  out.close();
    }
}
```

Database Connectivity

Java Database Connectivity

- JDBC stands for Java Database Connectivity.
- JDBC is a Java API to connect and execute the query with the database.
- The JDBC API defines interfaces and classes for writing databases applications in Java by making database connection.



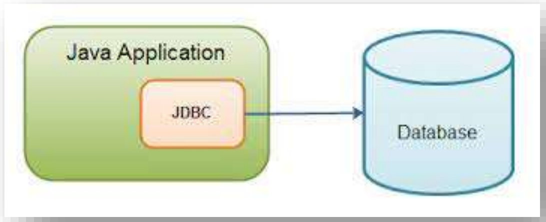
Database

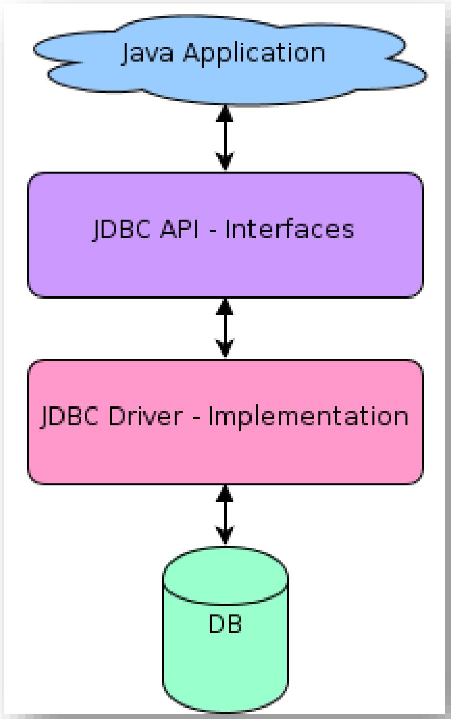
- Database is a set of files containing application data.

- This data needs to be inserted, deleted, updated, extracted for any valid reason.
  - You can write programs to perform all such actions
  - You can use readymade database management software like Oracle and MySQL.

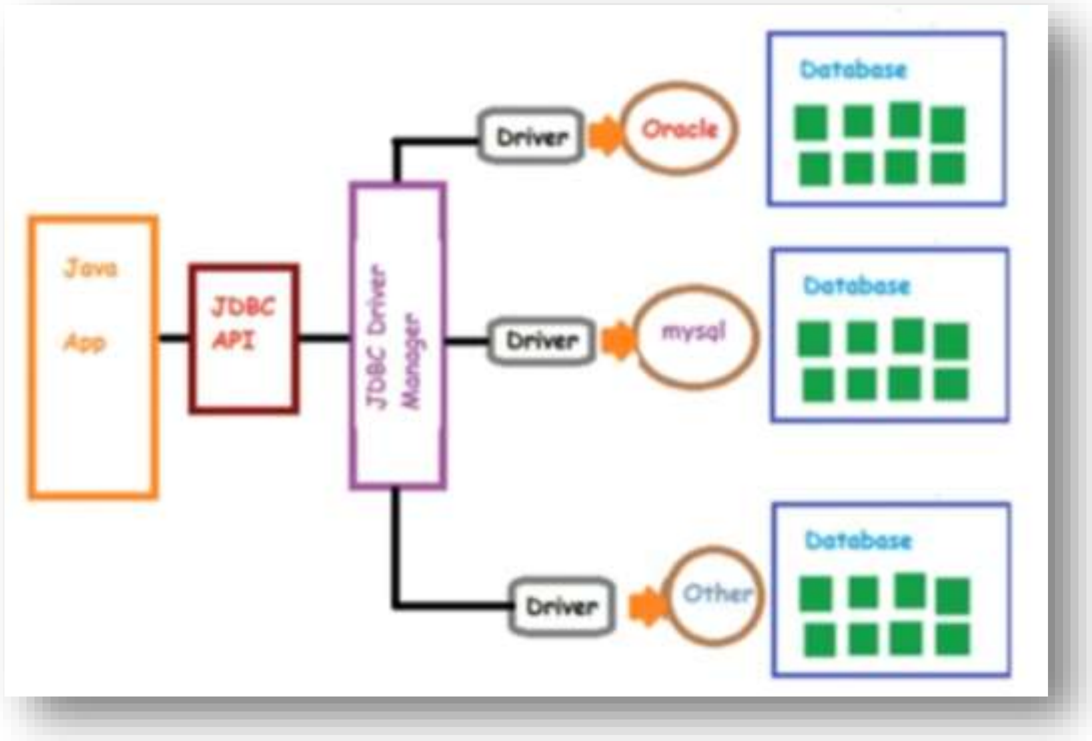
- A Database Management Software or DBMS is used for storing, manipulating, and managing data in a database environment. Users can construct their own databases using a DBMS to satisfy their business requirements.

Overview


Java Application
JDBC
Database


Java Application
JDBC API - Interfaces
JDBC Driver - Implementation
DB

JDBC Architecture



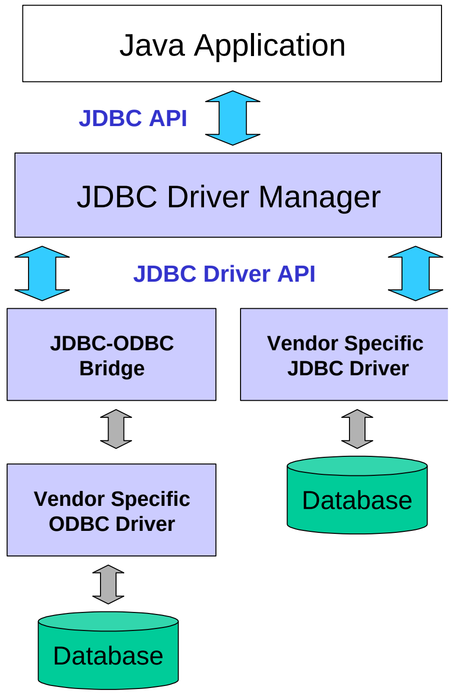
Why Should We Use JDBC

- Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

- We can use JDBC API to handle database using Java program and can perform the following activities:
  – Connect to the database
  – Execute queries and update statements to the database
  – Retrieve the result received from the database.

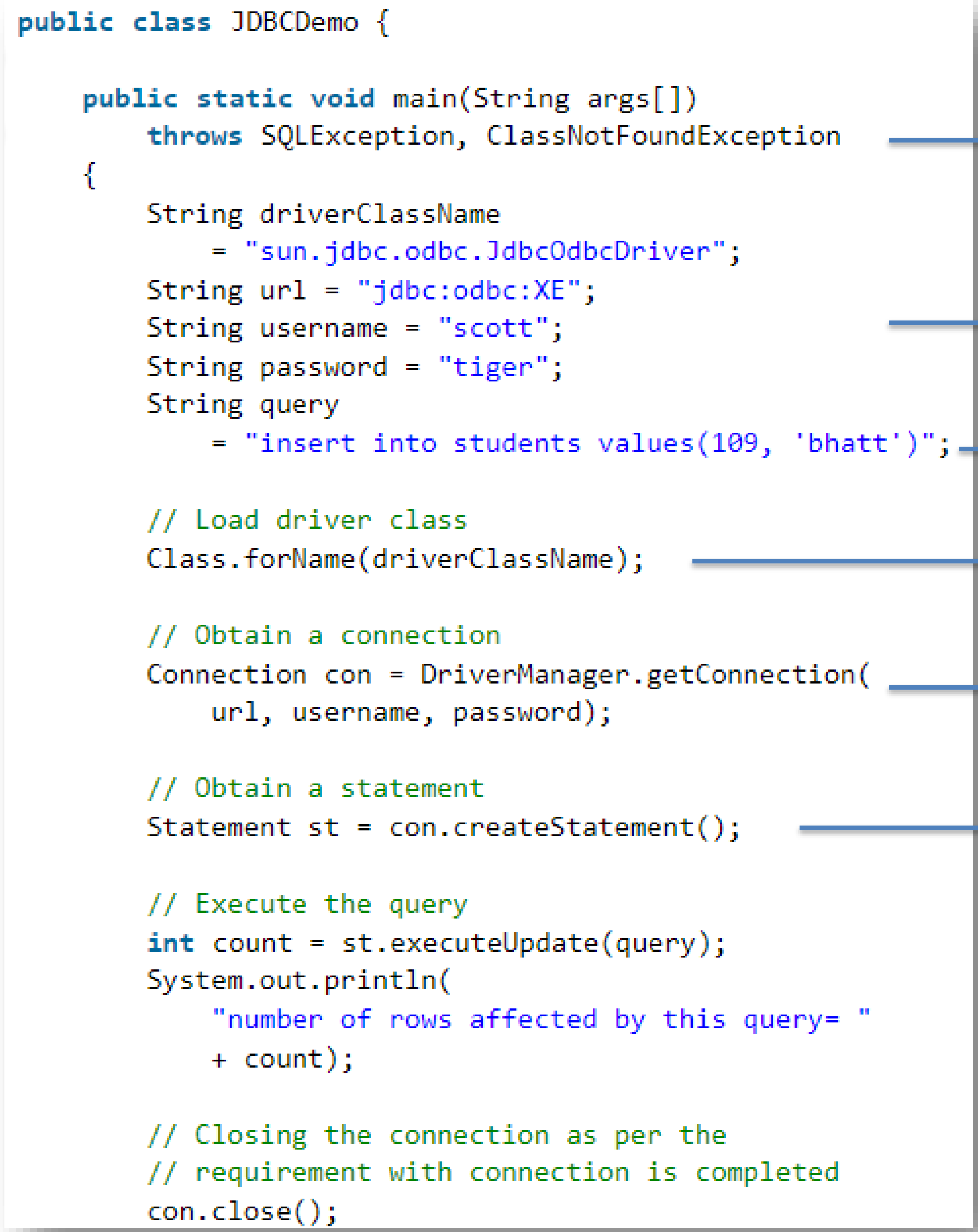
JDBC Drivers

- JDBC consists of two parts:
  - JDBC API, a purely Java-based API
  - JDBC Driver Manager, which communicates with vendor-specific drivers that perform the real communication with the database.



Basic steps to use a database in Java

1. Establish a **connection**
2. Create JDBC **Statements**
3. Execute **SQL** Statements
4. GET **ResultSet**
5. **Close** connections



```
public class JDBCDemo {

    public static void main(String args[])
        throws SQLException, ClassNotFoundException
    {
        String driverClassName
            = "sun.jdbc.odbc.JdbcOdbcDriver";
        String url = "jdbc:odbc:XE";
        String username = "scott";
        String password = "tiger";
        String query
            = "insert into students values(109, 'bhatt')";

        // Load driver class
        Class.forName(driverClassName);

        // Obtain a connection
        Connection con = DriverManager.getConnection(
            url, username, password);

        // Obtain a statement
        Statement st = con.createStatement();

        // Execute the query
        int count = st.executeUpdate(query);
        System.out.println(
            "number of rows affected by this query= "
            + count);

        // Closing the connection as per the
        // requirement with connection is completed
        con.close();
```

Handing SQL Exception

Setting DB Credentials

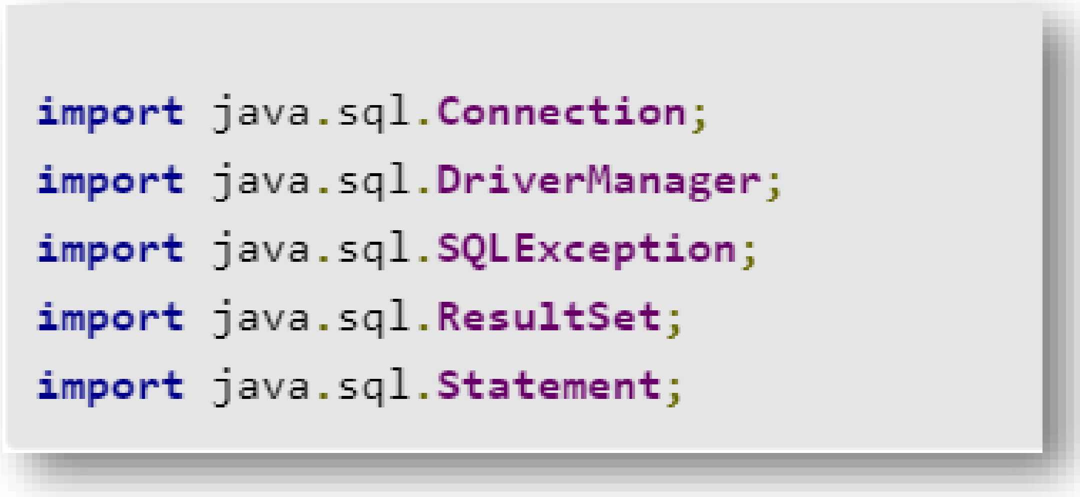
CRUD Query

Load driver

Establish Connection

Execute queries with the database

JDBC imports



```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.ResultSet;
import java.sql.Statement;
```

1. **Connection** represents the connection to the database.
2. **DriverManager** obtains the connection to the database.
3. **SQLException** handles SQL errors between the Java application and the database.
4. **ResultSet** and **Statement** model the data result sets and SQL statements.

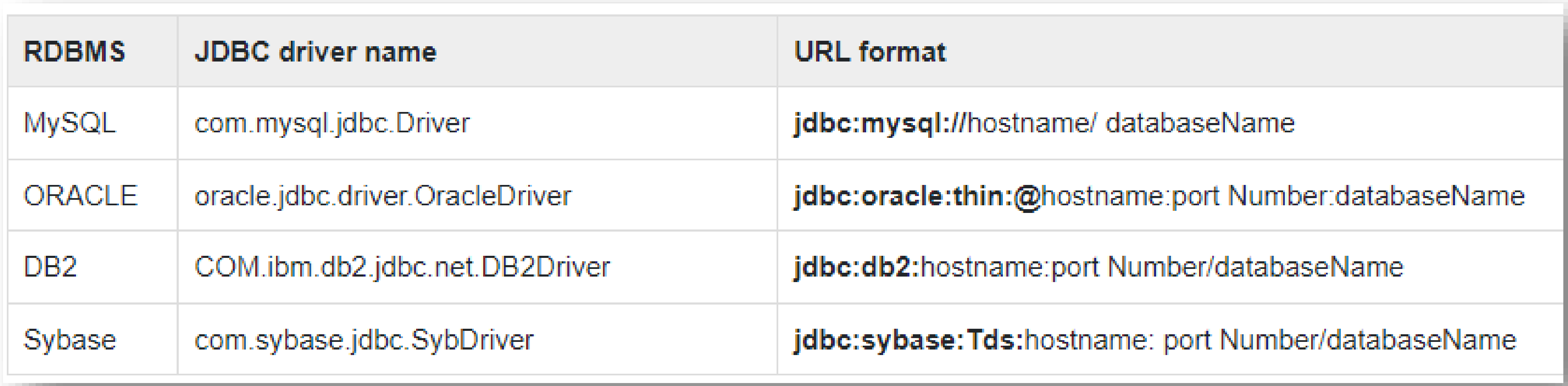
DriverManager class

- The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.

- The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

Connection interface

- Connection interface resides in **java.sql** package and it represents a session with a specific database you are connecting to.



RDBMS	JDBC driver name	URL format
MySQL	com.mysql.jdbc.Driver	**jdbc:mysql:**//hostname/ databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	**jdbc:oracle:thin:@**hostname:port Number:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	**jdbc:db2:**hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	**jdbc:sybase:Tds:**hostname: port Number/databaseName

Statement interface

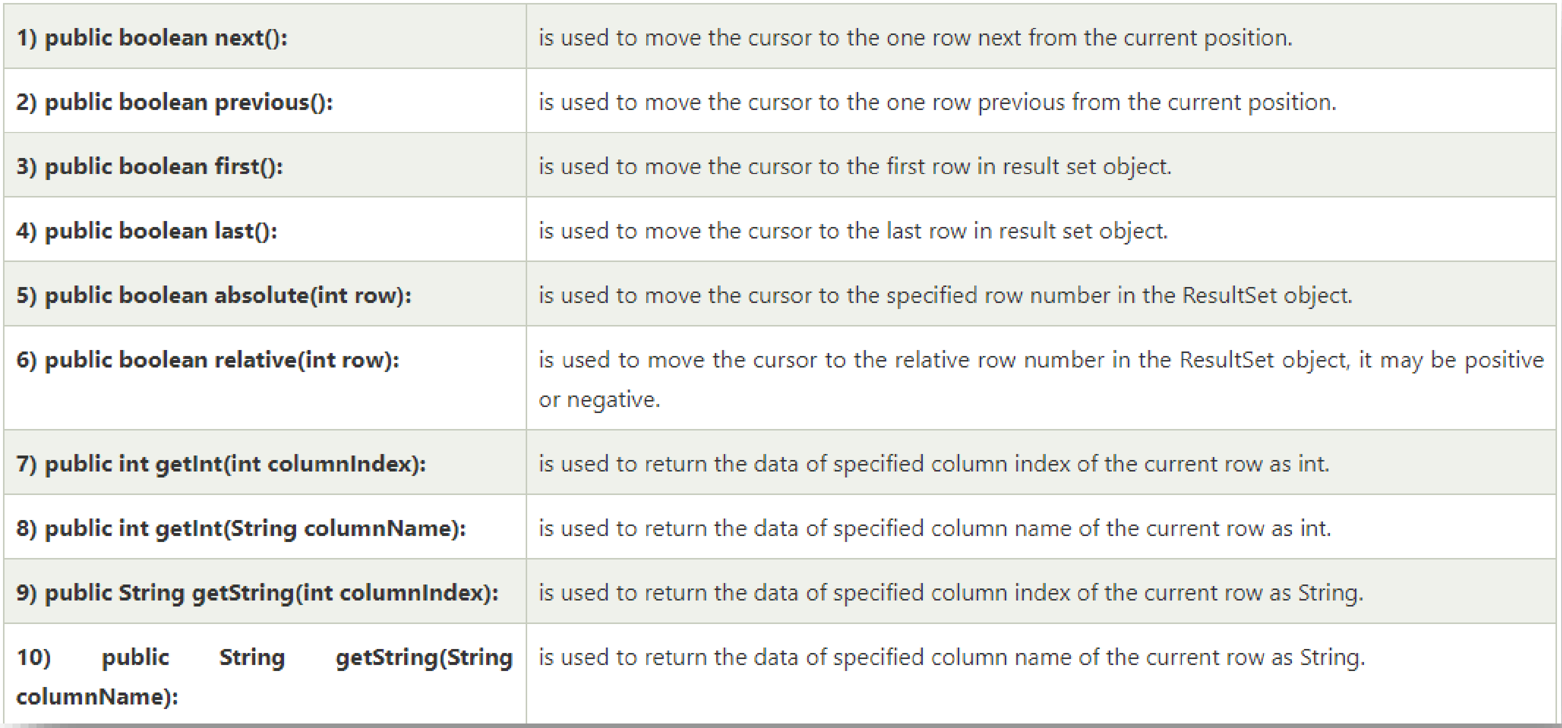
- The Statement interface *provides methods to execute queries* with the database.

- The important methods of Statement interface are as follows:

1. **public ResultSet executeQuery(String sql):** is used to execute SELECT query. It returns the object of ResultSet.
2. **public int executeUpdate(String sql):** is used to execute specified query, it may be create, drop, insert, update, delete etc.
3. **public boolean execute(String sql):** is used to execute queries that may return multiple results.
4. **public int[] executeBatch():** is used to execute batch of commands.

ResultSet interface

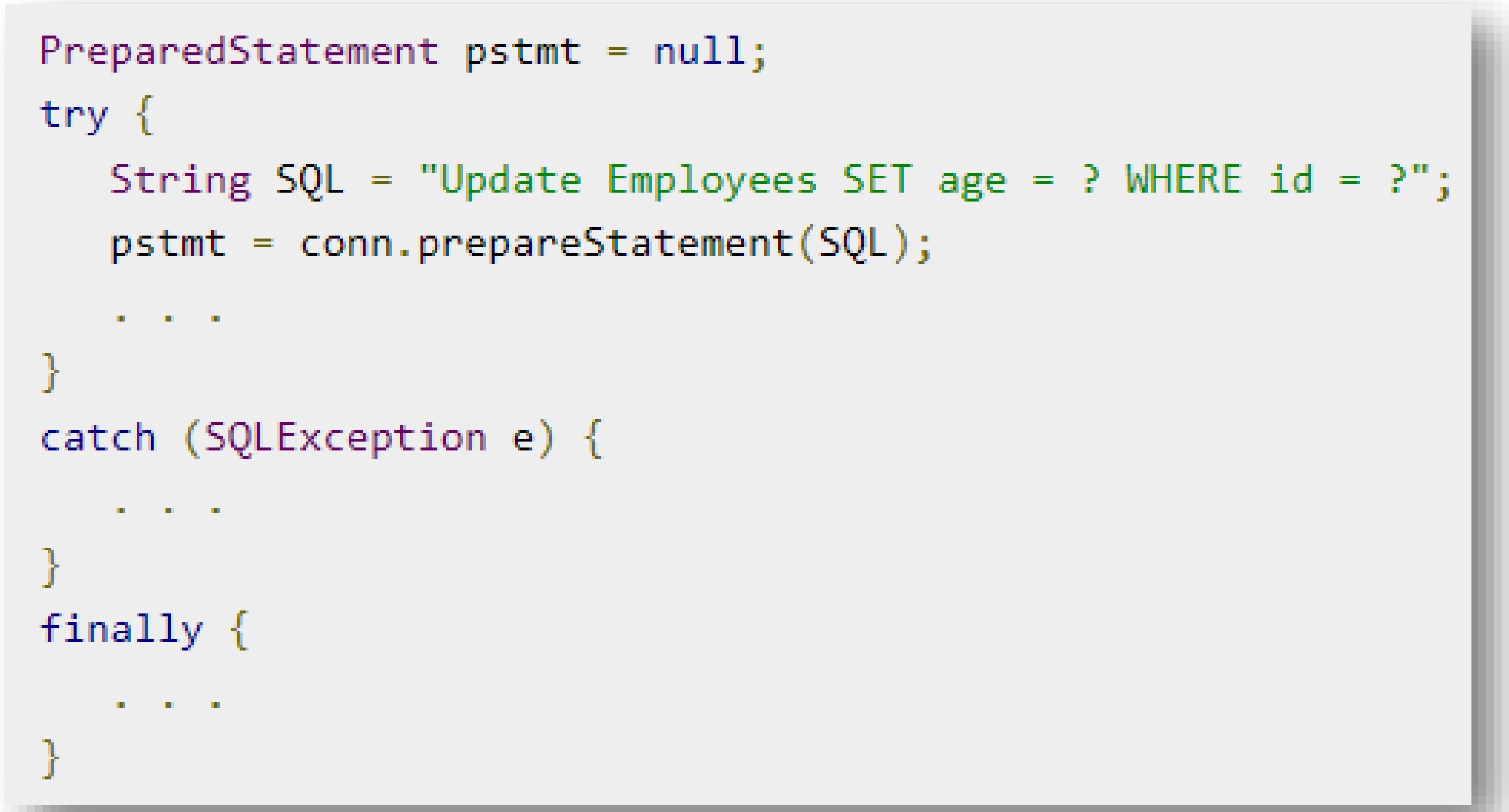
- The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.



1) public boolean next():	is used to move the cursor to the one row next from the current position.
2) public boolean previous():	is used to move the cursor to the one row previous from the current position.
3) public boolean first():	is used to move the cursor to the first row in result set object.
4) public boolean last():	is used to move the cursor to the last row in result set object.
5) public boolean absolute(int row):	is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean relative(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
7) public int getInt(int columnIndex):	is used to return the data of specified column index of the current row as int.
8) public int getInt(String columnName):	is used to return the data of specified column name of the current row as int.
9) public String getString(int columnIndex):	is used to return the data of specified column index of the current row as String.
10) public String getString(String columnName):	is used to return the data of specified column name of the current row as String.

PreparedStatement interface

- The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.



```
PreparedStatement pstmt = null;
try {
   String SQL = "Update Employees SET age = ? WHERE id = ?";
   pstmt = conn.prepareStatement(SQL);
   . . .
}
catch (SQLException e) {
   . . .
}
finally {
   . . .
}
```

Oracle Java Connectivity Demo

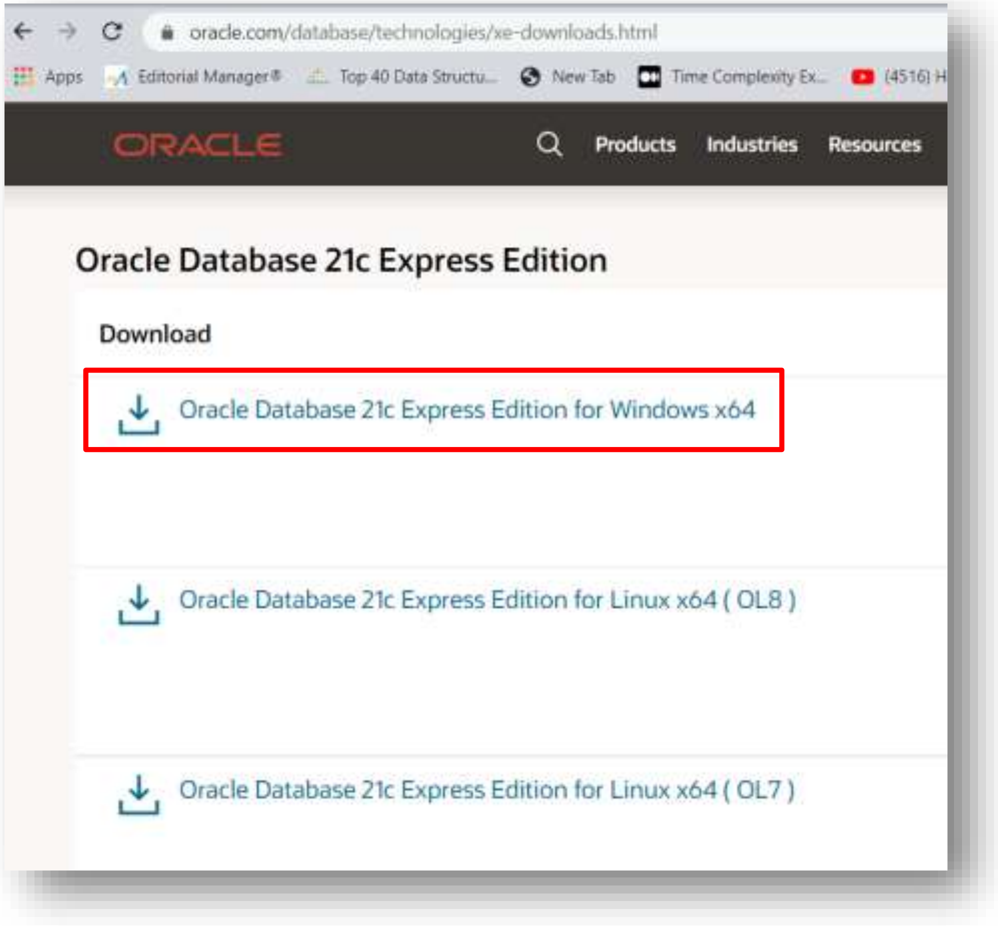
quest lab

Requirements

- Eclipse
- JDK
- Oracle Setup
- Oracle JDBC Driver

Install Oracle

- Install https://www.oracle.com/database/technologies/xe-downloads.html


oracle.com/database/technologies/xe-downloads.html

Apps · Editorial Manager® · Top 40 Data Structu... · New Tab · Time Complexity Ex... · (4516) H

ORACLE · Products · Industries · Resources

Oracle Database 21c Express Edition

Download

Oracle Database 21c Express Edition for Windows x64

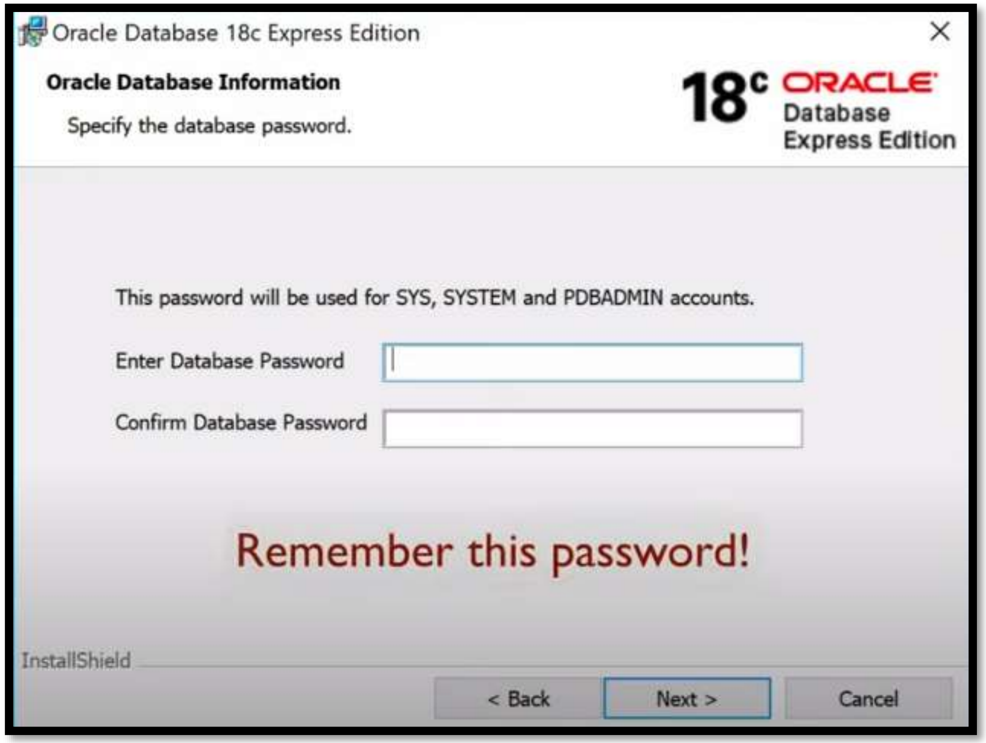
Oracle Database 21c Express Edition for Linux x64 (OL8)

Oracle Database 21c Express Edition for Linux x64 (OL7)

Prerequisites

- Oracle JDBC Driver
  https://www.oracle.com/database/technologies/appdev/jdbc-downloads.html

Important to remember password


Oracle Database 18c Express Edition

Oracle Database Information

Specify the database password.

This password will be used for SYS, SYSTEM and PDBADMIN accounts.

Enter Database Password

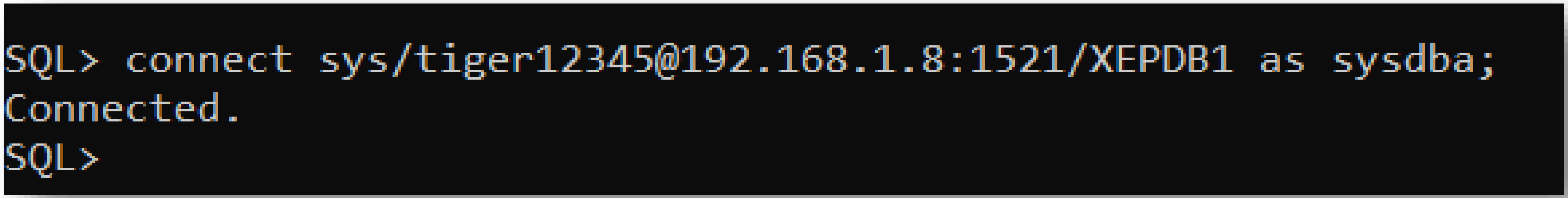
Confirm Database Password

Remember this password!

Connect to database

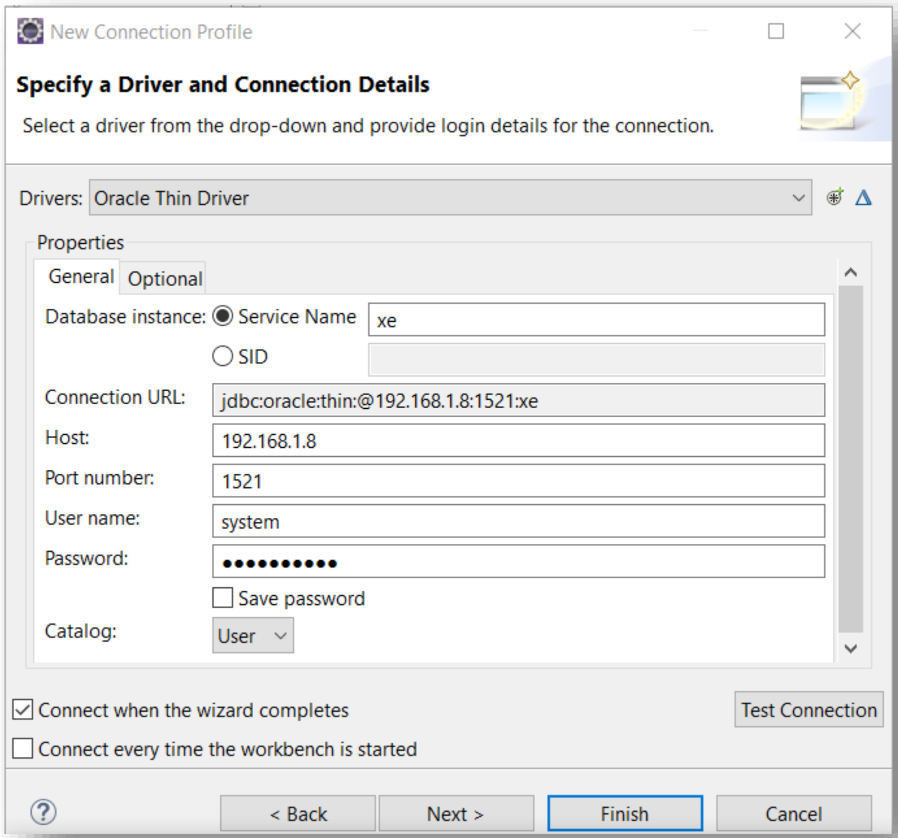
Open SQL Plus

connect sys/[**password set during oracle installation**]@localhost:1521/XE as sysdba;



```
SQL> connect sys/tiger12345@192.168.1.8:1521/XEPDB1 as sysdba;
Connected.
SQL>
```

Open Eclipse-> open perspective -> Database Development -> New Connection Profile -> Oracle -> New Driver Definition -> Oracle Thin Driver -> Jar List -> provide OJDBC.jar



New Connection Profile

Specify a Driver and Connection Details

Select a driver from the drop-down and provide login details for the connection.

Drivers: Oracle Thin Driver

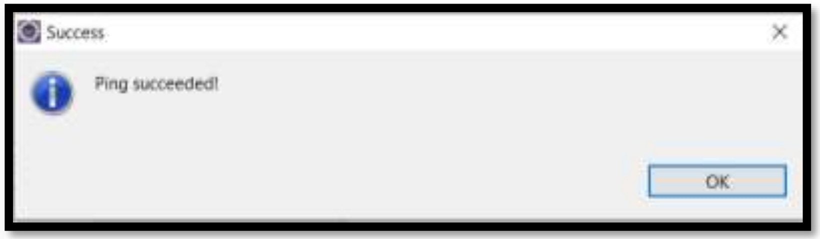
Properties

General | Optional

Database instance:	[✓ Service Name]	xe
	SID	
Connection URL:	jdbc:oracle:thin:@192.168.1.8:1521:xe	
Host:	192.168.1.8	
Port number:	1521	
User name:	system	
Password:	●●●●●●●●●●●	
	☐ Save password	
Catalog:	User	

☑ Connect when the wizard completes

☐ Connect every time the workbench is started

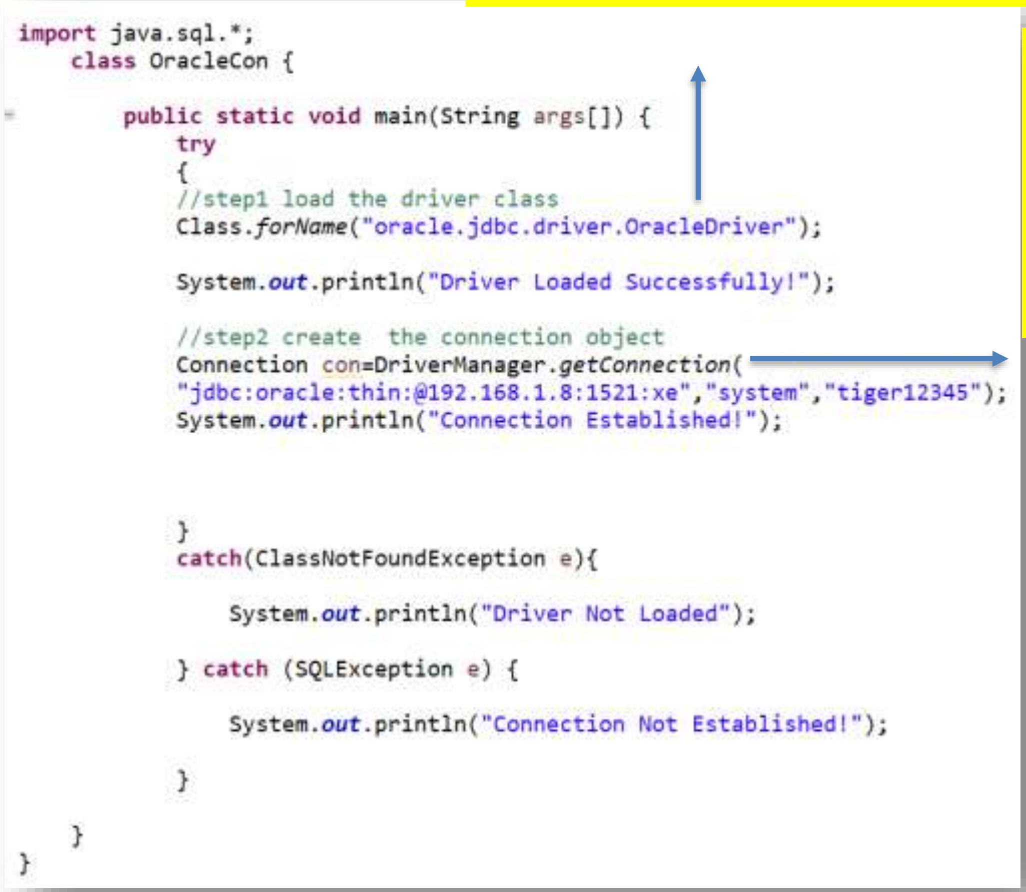


Success

Ping succeeded!

Connecting to the database

The driver class for the oracle database is **oracle.jdbc.driver.OracleDriver**.

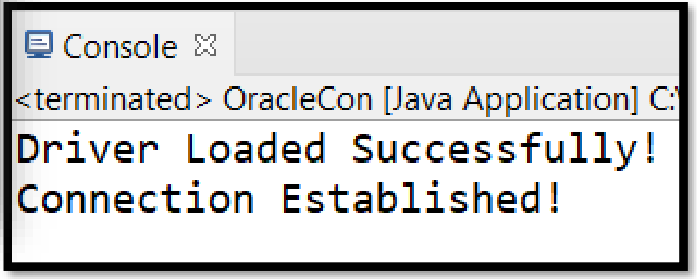


```java
import java.sql.*;
    class OracleCon {

        public static void main(String args[]) {
            try
            {
            //step1 load the driver class
            Class.forName("oracle.jdbc.driver.OracleDriver");

            System.out.println("Driver Loaded Successfully!");

            //step2 create  the connection object
            Connection con=DriverManager.getConnection(
            "jdbc:oracle:thin:@192.168.1.8:1521:xe","system","tiger12345");
            System.out.println("Connection Established!");


            }
            catch(ClassNotFoundException e){

                System.out.println("Driver Not Loaded");

            } catch (SQLException e) {

                System.out.println("Connection Not Established!");

            }
        }
}
```

where **jdbc** is the API, **oracle** is the database, **thin** is the driver, **IP address**, **1521** is the port number and **XE** is the Oracle service name.

The default username for the oracle database is **system**.

password given by the user at the time of installing the oracle database.

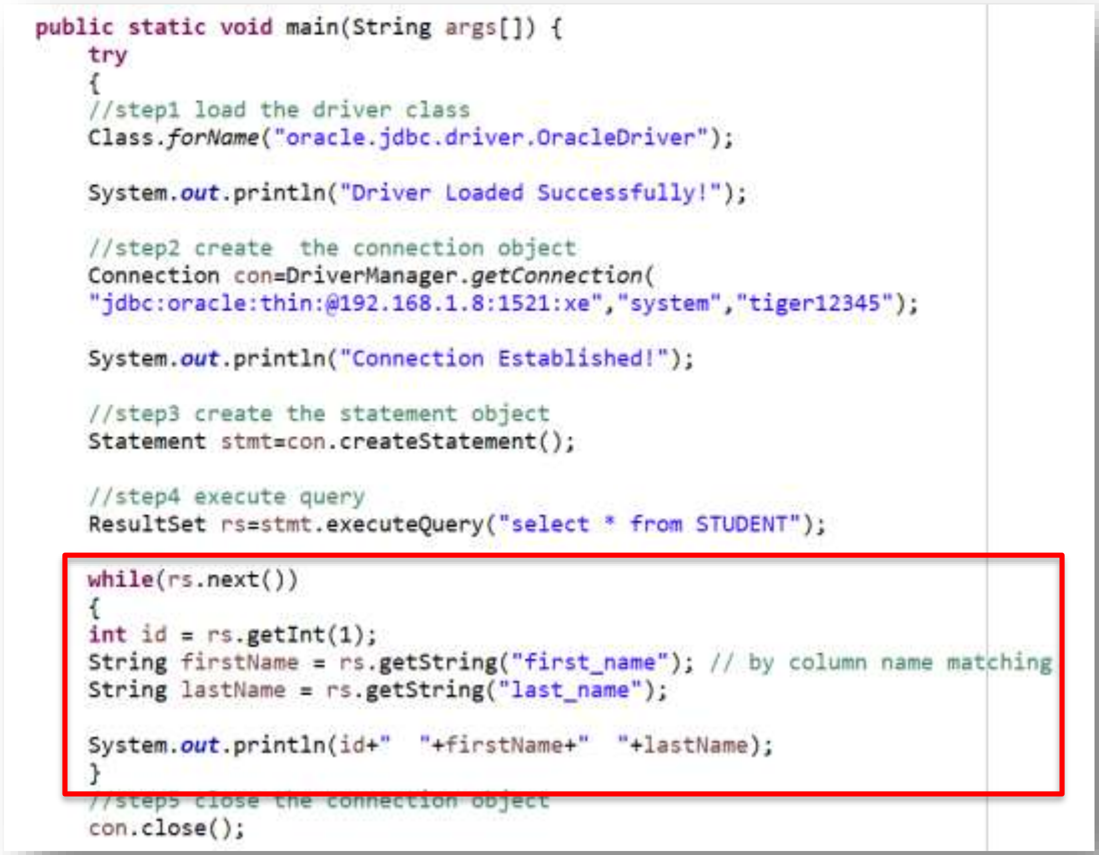


```
Console
<terminated> OracleCon [Java Application] C:\
Driver Loaded Successfully!
Connection Established!
```

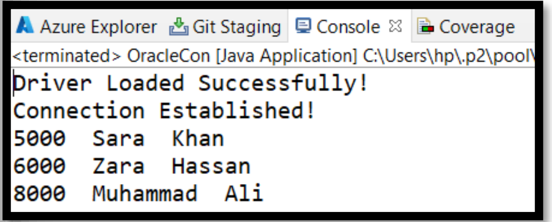
CRUD Operations

quest lab

Retrieve Data from Database

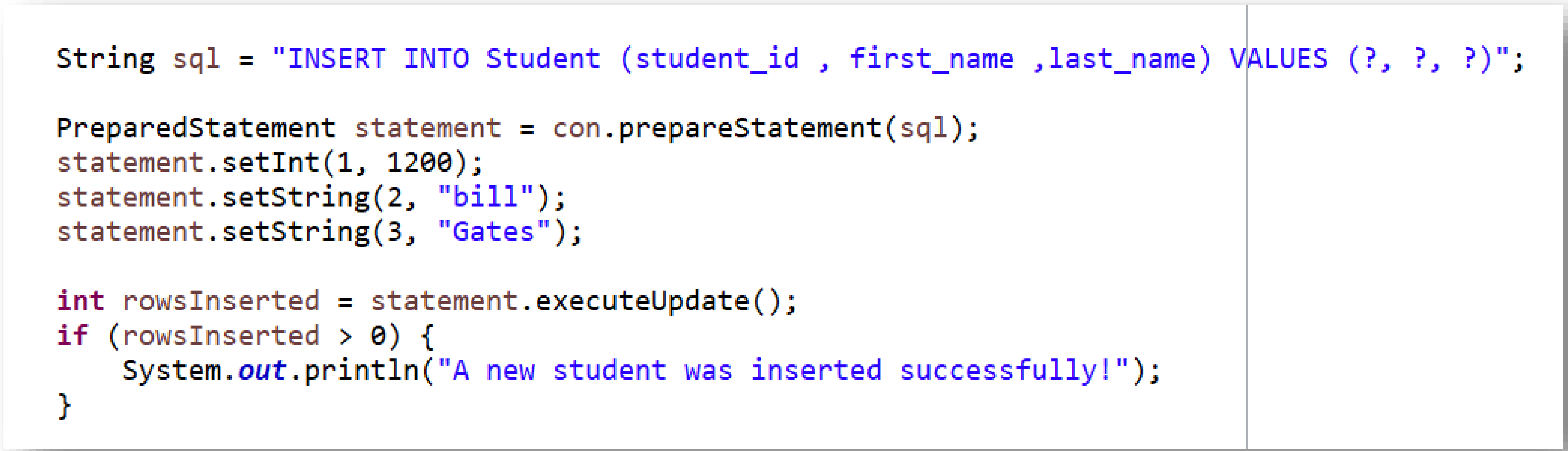


```
public static void main(String args[]) {
    try
    {
    //step1 load the driver class
    Class.forName("oracle.jdbc.driver.OracleDriver");

    System.out.println("Driver Loaded Successfully!");

    //step2 create  the connection object
    Connection con=DriverManager.getConnection(
    "jdbc:oracle:thin:@192.168.1.8:1521:xe","system","tiger12345");

    System.out.println("Connection Established!");

    //step3 create the statement object
    Statement stmt=con.createStatement();

    //step4 execute query
    ResultSet rs=stmt.executeQuery("select * from STUDENT");

    while(rs.next())
    {
    int id = rs.getInt(1);
    String firstName = rs.getString("first_name"); // by column name matching
    String lastName = rs.getString("last_name");

    System.out.println(id+"  "+firstName+"  "+lastName);
    }
    //step5 close the connection object
    con.close();
```

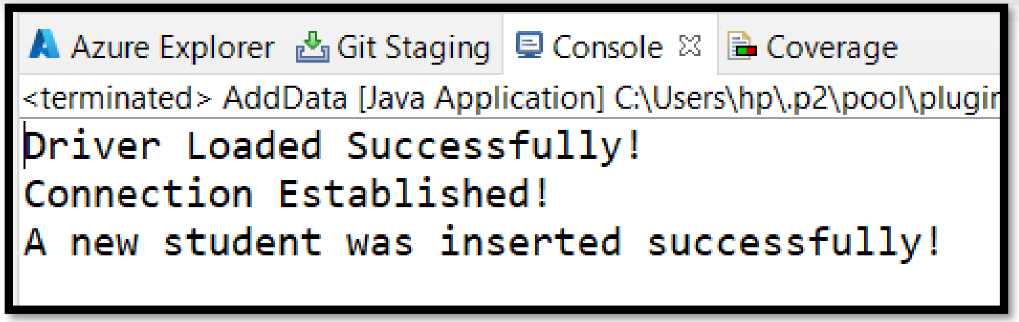


```
<terminated> OracleCon [Java Application] C:\Users\hp\.p2\pool\
Driver Loaded Successfully!
Connection Established!
5000  Sara  Khan
6000  Zara  Hassan
8000  Muhammad  Ali
```

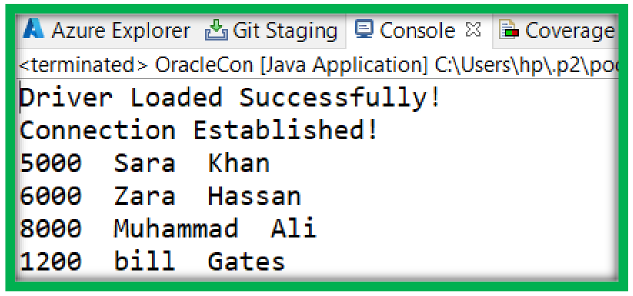
INSERT Statement



```java
String sql = "INSERT INTO Student (student_id , first_name ,last_name) VALUES (?, ?, ?)";

PreparedStatement statement = con.prepareStatement(sql);
statement.setInt(1, 1200);
statement.setString(2, "bill");
statement.setString(3, "Gates");

int rowsInserted = statement.executeUpdate();
if (rowsInserted > 0) {
    System.out.println("A new student was inserted successfully!");
}
```

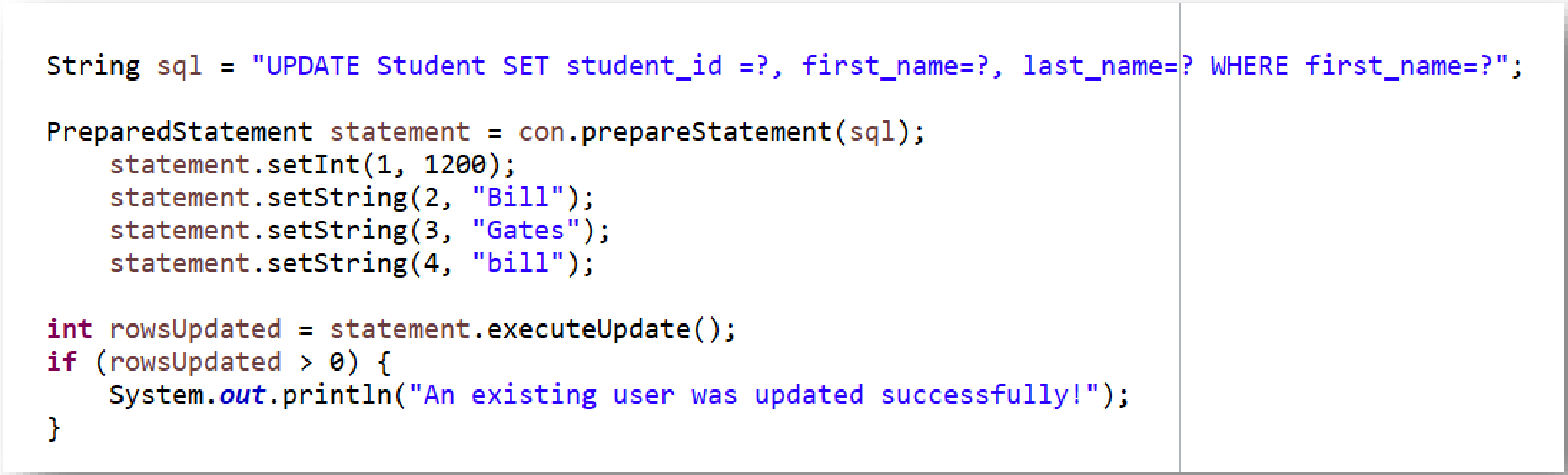


```
<terminated> AddData [Java Application] C:\Users\hp\.p2\pool\plugir
Driver Loaded Successfully!
Connection Established!
A new student was inserted successfully!
```

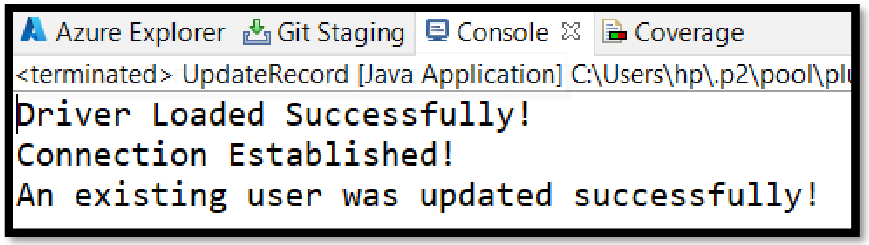


```
<terminated> OracleCon [Java Application] C:\Users\hp\.p2\poo
Driver Loaded Successfully!
Connection Established!
5000  Sara  Khan
6000  Zara  Hassan
8000  Muhammad  Ali
1200  bill  Gates
```

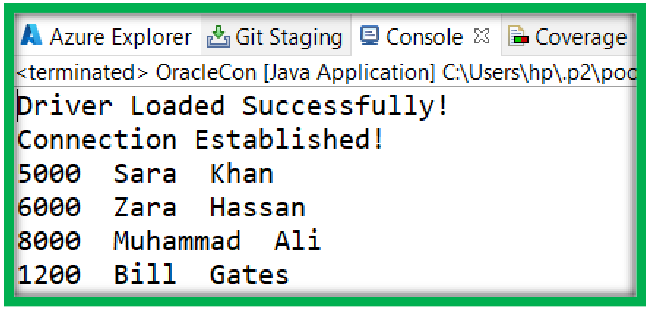
UPDATE Statement



```java
String sql = "UPDATE Student SET student_id =?, first_name=?, last_name=? WHERE first_name=?";

PreparedStatement statement = con.prepareStatement(sql);
    statement.setInt(1, 1200);
    statement.setString(2, "Bill");
    statement.setString(3, "Gates");
    statement.setString(4, "bill");

int rowsUpdated = statement.executeUpdate();
if (rowsUpdated > 0) {
    System.out.println("An existing user was updated successfully!");
}
```

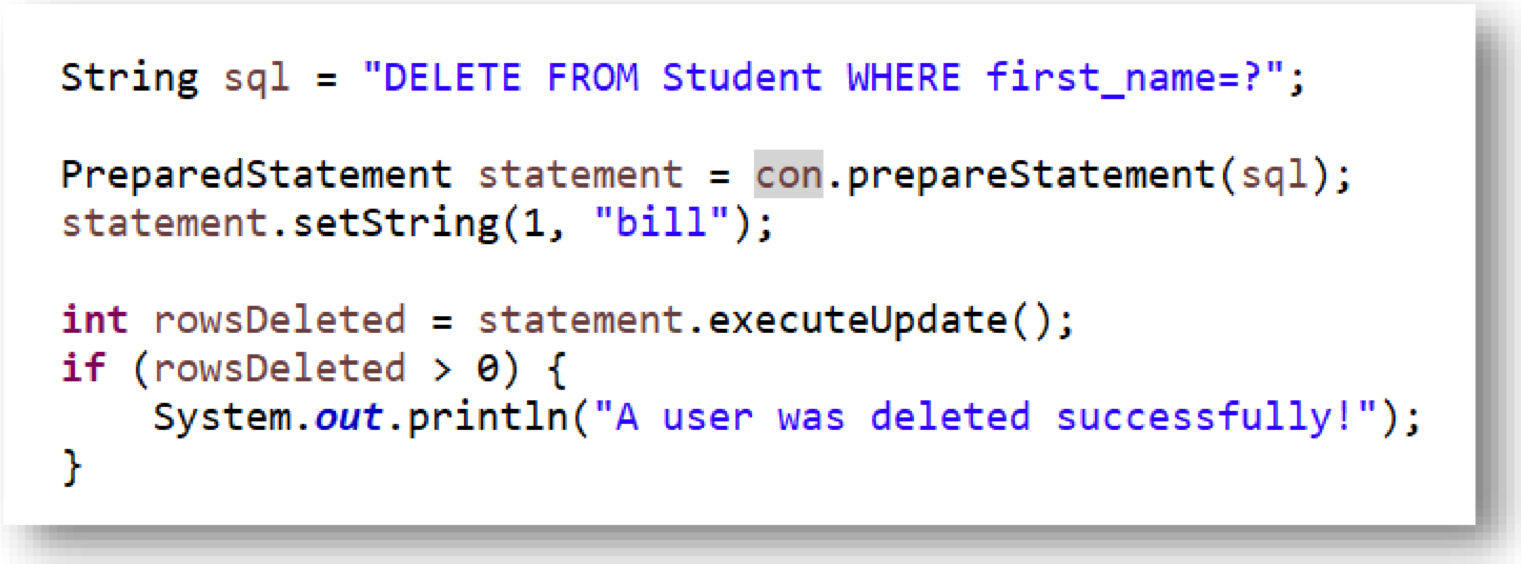


```
<terminated> UpdateRecord [Java Application] C:\Users\hp\.p2\pool\pl
Driver Loaded Successfully!
Connection Established!
An existing user was updated successfully!
```

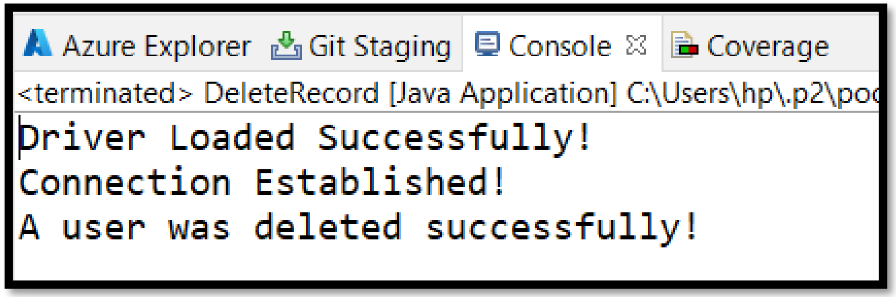


```
<terminated> OracleCon [Java Application] C:\Users\hp\.p2\poo
Driver Loaded Successfully!
Connection Established!
5000  Sara  Khan
6000  Zara  Hassan
8000  Muhammad  Ali
1200  Bill  Gates
```

DELETE Statement

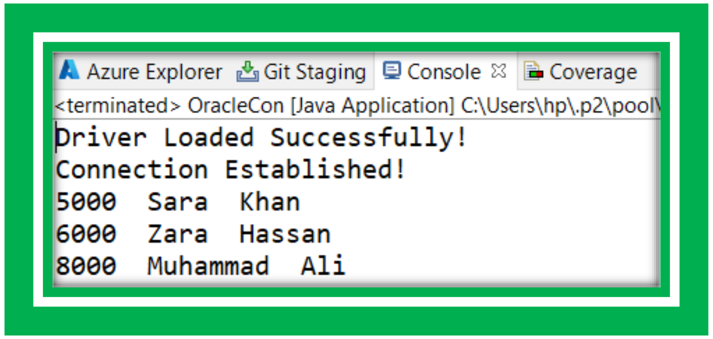


```
String sql = "DELETE FROM Student WHERE first_name=?";

PreparedStatement statement = con.prepareStatement(sql);
statement.setString(1, "bill");

int rowsDeleted = statement.executeUpdate();
if (rowsDeleted > 0) {
    System.out.println("A user was deleted successfully!");
}
```



Azure Explorer | Git Staging | Console | Coverage

```
<terminated> DeleteRecord [Java Application] C:\Users\hp\.p2\poo
Driver Loaded Successfully!
Connection Established!
A user was deleted successfully!
```



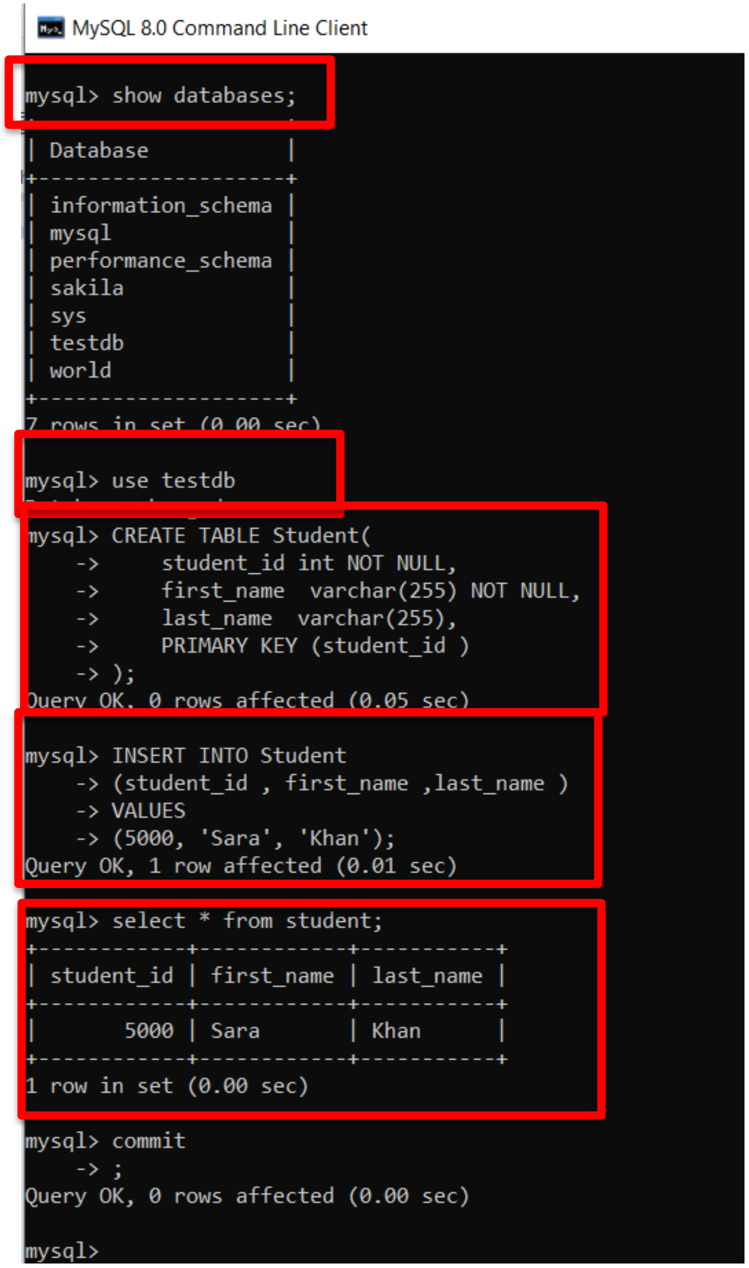
Azure Explorer | Git Staging | Console | Coverage

```
<terminated> OracleCon [Java Application] C:\Users\hp\.p2\pool\
Driver Loaded Successfully!
Connection Established!
5000  Sara  Khan
6000  Zara  Hassan
8000  Muhammad  Ali
```

MySQL Database

- Connect to MySQL database in Eclipse IDE using Database Development perspective:
  - Download MySQL https://dev.mysql.com/downloads/installer/
  - Download MySQL JDBC driver https://dev.mysql.com/downloads/connector/j/
  - Create Database and Table in MySQL
  - Execute SQL Statements

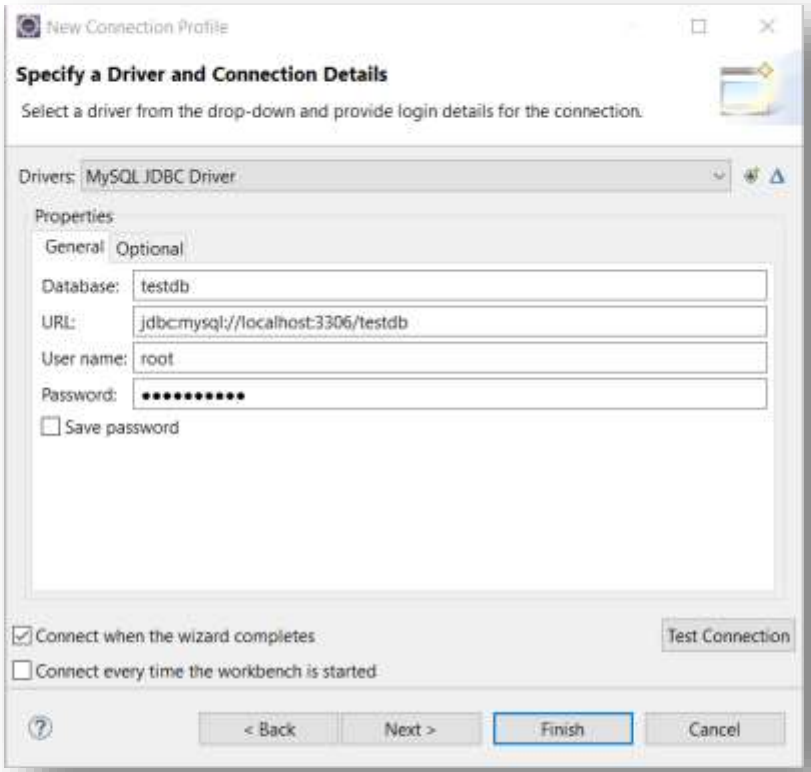
MySQL Command Line Activities



MySQL 8.0 Command Line Client

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sakila             |
| sys                |
| testdb             |
| world              |
+--------------------+
7 rows in set (0.00 sec)

mysql> use testdb
Database changed
mysql> CREATE TABLE Student(
    ->     student_id int NOT NULL,
    ->     first_name  varchar(255) NOT NULL,
    ->     last_name  varchar(255),
    ->     PRIMARY KEY (student_id )
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO Student
    -> (student_id , first_name ,last_name )
    -> VALUES
    -> (5000, 'Sara', 'Khan');
Query OK, 1 row affected (0.01 sec)

mysql> select * from student;
+------------+------------+-----------+
| student_id | first_name | last_name |
+------------+------------+-----------+
|       5000 | Sara       | Khan      |
+------------+------------+-----------+
1 row in set (0.00 sec)

mysql> commit
    -> ;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Database Development perspective in Eclipse



New Connection Profile

Specify a Driver and Connection Details

Select a driver from the drop-down and provide login details for the connection.

Drivers: MySQL JDBC Driver

Properties

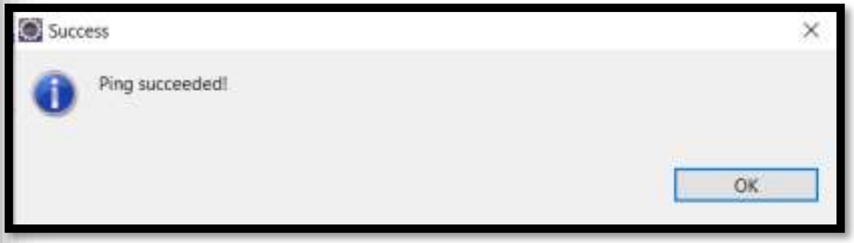
General | Optional

Database:	testdb
URL:	jdbc:mysql://localhost:3306/testdb
User name:	root
Password:	●●●●●●●●●●●

☐ Save password

☑ Connect when the wizard completes

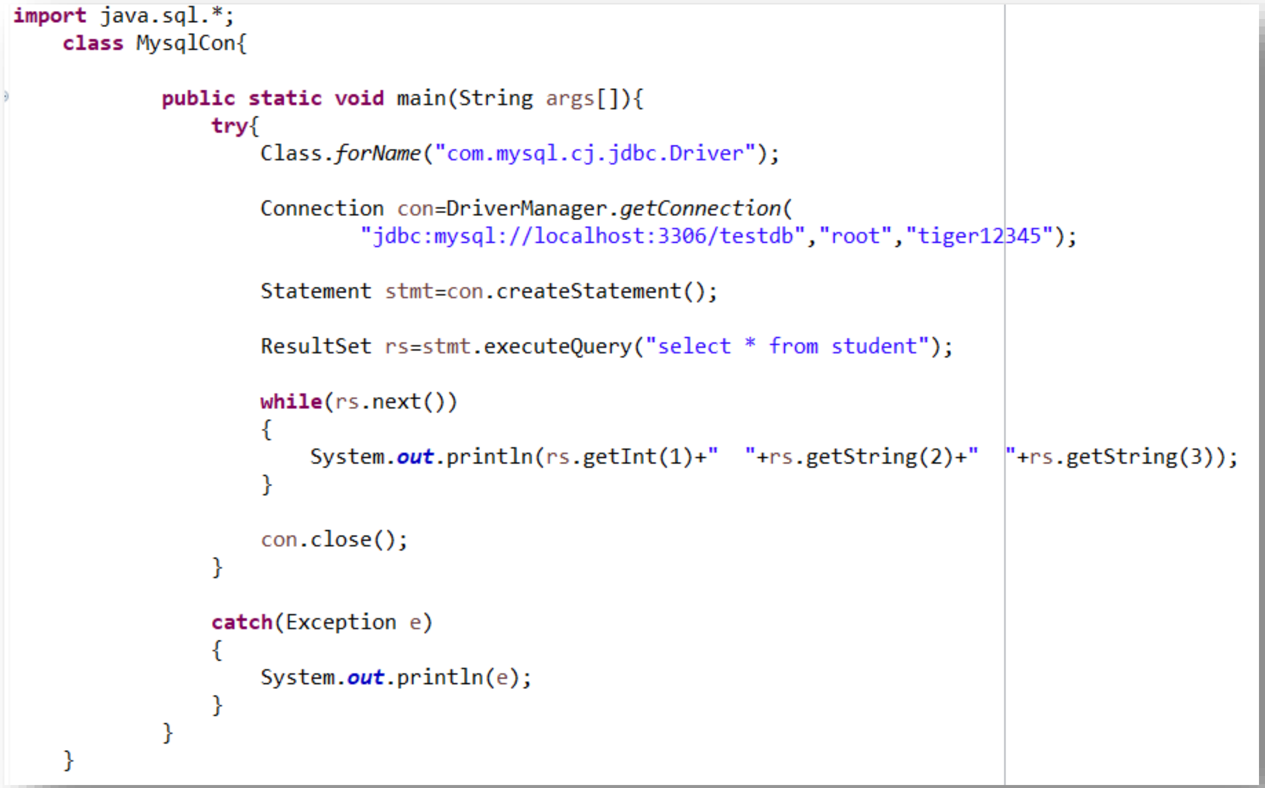
☐ Connect every time the workbench is started



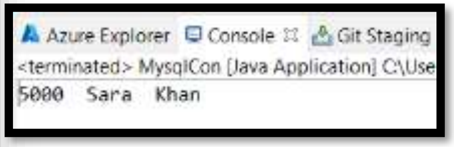
Success

Ping succeeded!

Connect Java Application with mysql database



```
import java.sql.*;
    class MysqlCon{

            public static void main(String args[]){
                try{
                    Class.forName("com.mysql.cj.jdbc.Driver");

                    Connection con=DriverManager.getConnection(
                            "jdbc:mysql://localhost:3306/testdb","root","tiger12345");

                    Statement stmt=con.createStatement();

                    ResultSet rs=stmt.executeQuery("select * from student");

                    while(rs.next())
                    {
                        System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
                    }

                    con.close();
                }

                catch(Exception e)
                {
                    System.out.println(e);
                }
            }
    }
```



```
<terminated> MysqlCon [Java Application] C:\Use
5000  Sara  Khan
```

Design for Change

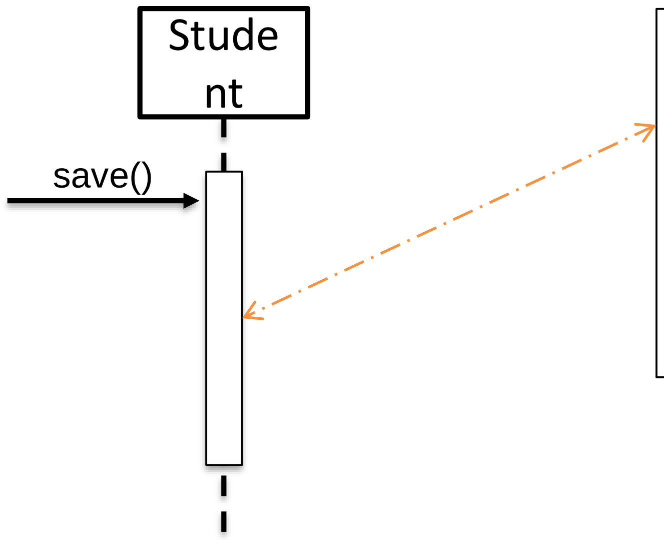
- Identify the functionality that may change (frequently?)
- Take special consideration in implementing such functionality in classes, so that the changing functionality will have minimized impact on the other parts
- Use the principles of Polymorphism to provide a **stable interface** of the potentially varying functionality.

Design for Change – Write to Interfaces

- The client classes are to be written in a way that they talk to the stable interface

- Example – Handling Persistence
  - Probability of change?

Example – Handling Persistence

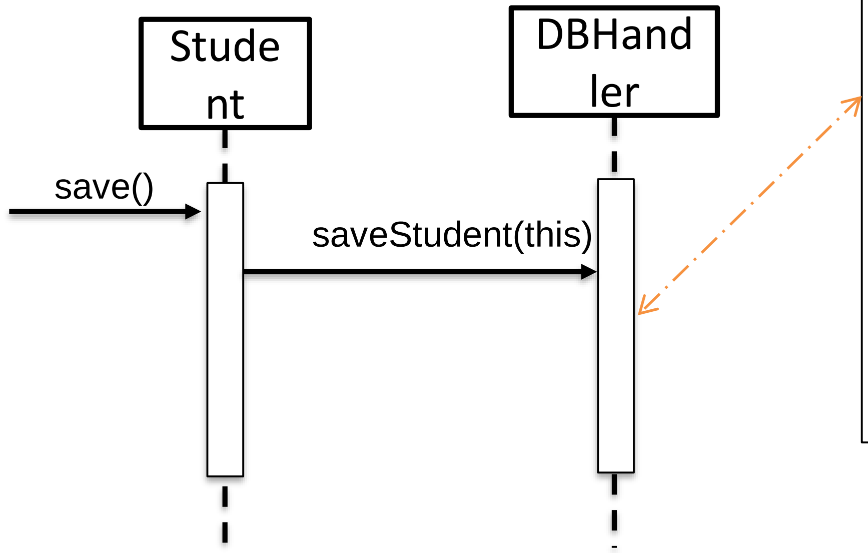


Option 1

- Write SQL Queries here

- Problem?

Example – Handling Persistence



Option 2

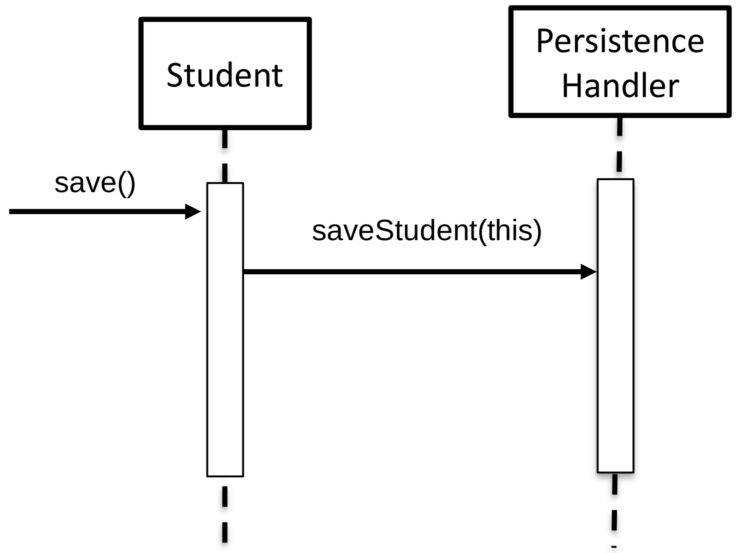
- Write SQL Queries here

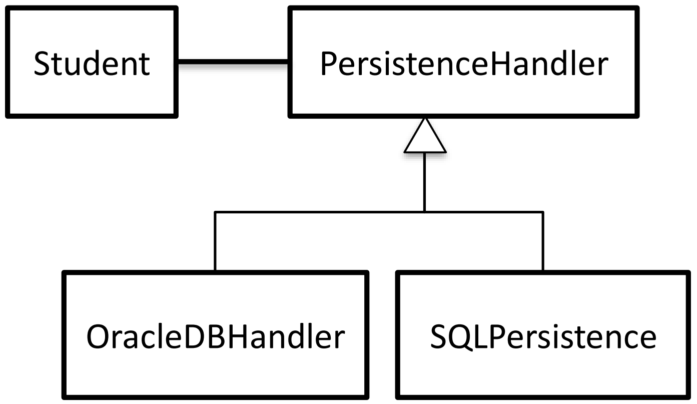
- Benefits over Option 1
- Problem?

Example – Write to Interfaces

Option 3
- Write to interfaces

- Benefits over other options
- Problem? When to initialize and how?





Implementation

//Student

```
Class Student{
   PersitenceHandler persHandler;

        void save(){
            persHandler.saveStudent(this);
                    }

        void setPersitenceHandler (PersitenceHandler ph)
                  {
              this.persHandler=ph;
                  }
```

Implementation

```
// PersistenceHandler
Class PersistenceHandler{
    abstract void saveStudent(Student s);
}
```

Implementation

```
class OracleDBHandler extends PersistenceHandler{

     void saveStudent(Student s){
          //connection
          //insert query formulation
          //execute query
    }
    }
```

Implementation

```
class SQLHandler extends PersistenceHandler{

     void saveStudent(Student s){
    //connection
    //insert query formulation
    //execute query
    }
    }
```

Implementation

Main

```
Void main()
{
    PersitenceHandler handler= new OracleHandler();

    University uni= new University();

    Uni. setPersitenceHandler(handler);
}
```

Task 02

- Add database in to your Account management system
- There should be a menu where you ask user where he wants to store his data. Following are the options
  - File
  - Oracle
  - MySQL