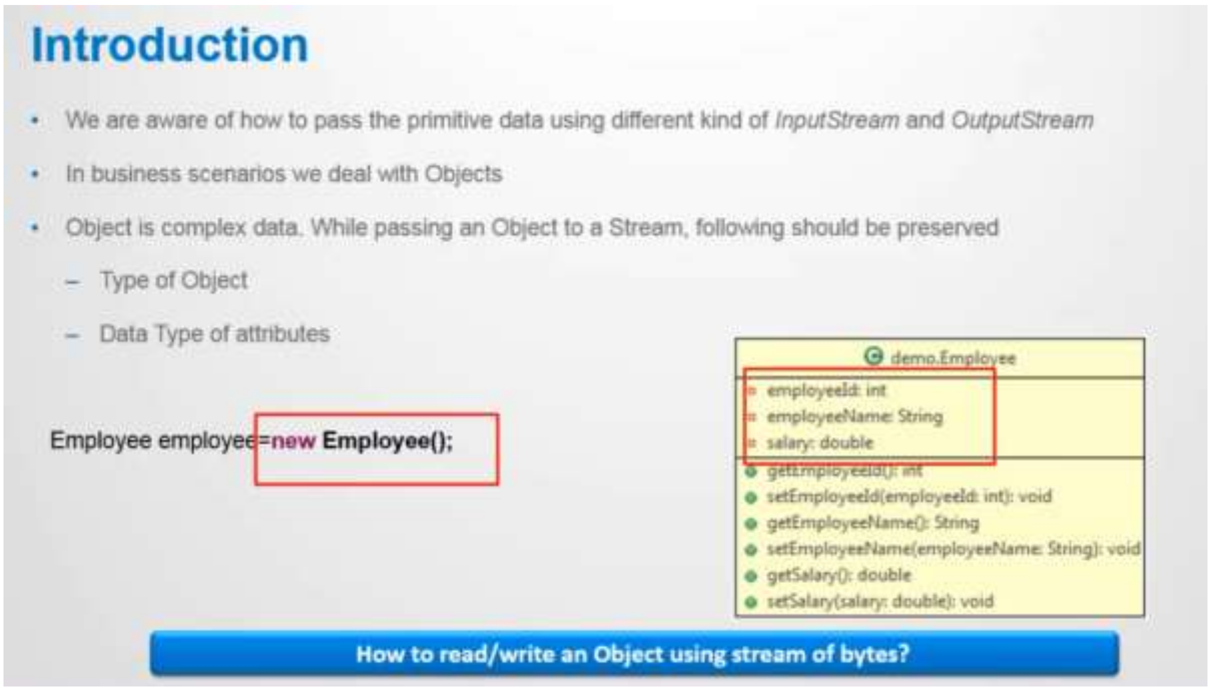
Serialization Deserialization:



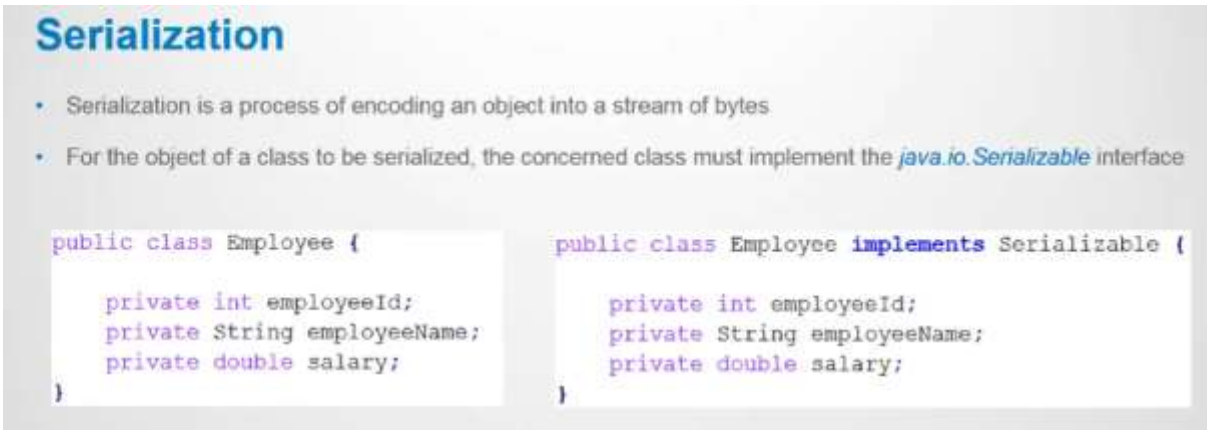
Introduction

- We are aware of how to pass the primitive data using different kind of *InputStream* and *OutputStream*
- In business scenarios we deal with Objects
- Object is complex data. While passing an Object to a Stream, following should be preserved
  - Type of Object
  - Data Type of attributes

```
Employee employee=new Employee();
```

```
demo.Employee
employeeId: int
employeeName: String
salary: double
getEmployeeId(): int
setEmployeeId(employeeId: int): void
getEmployeeName(): String
setEmployeeName(employeeName: String): void
getSalary(): double
setSalary(salary: double): void
```

How to read/write an Object using stream of bytes?



Serialization

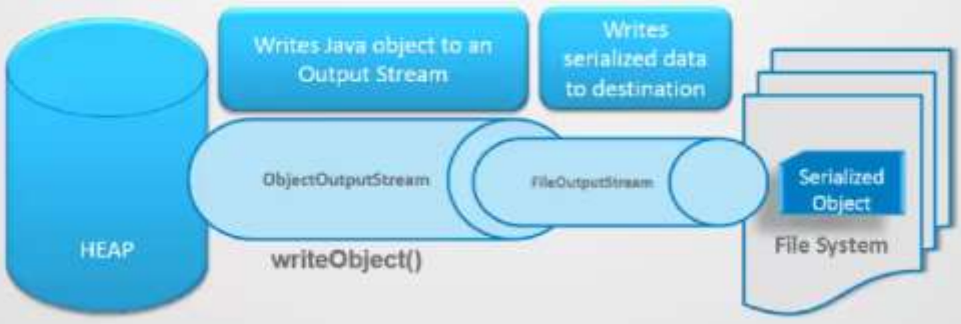
- Serialization is a process of encoding an object into a stream of bytes
- For the object of a class to be serialized, the concerned class must implement the *java.io.Serializable* interface

```
public class Employee {

    private int employeeId;
    private String employeeName;
    private double salary;
}
```

```
public class Employee implements Serializable {

    private int employeeId;
    private String employeeName;
    private double salary;
}
```

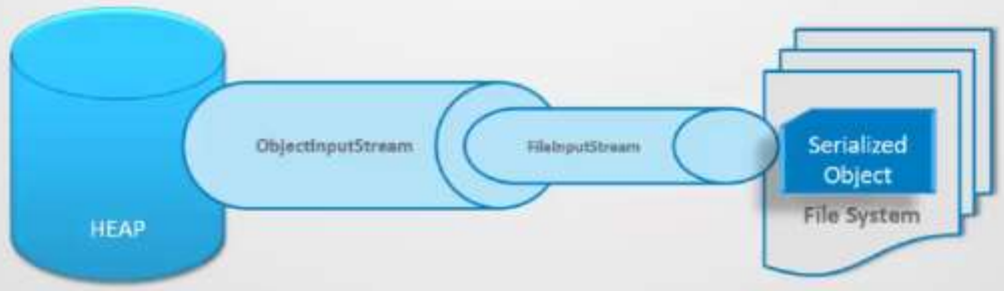
Serialization

- Serialization is a process of encoding an object into a stream of bytes
- It is achieved using ObjectOutputStream



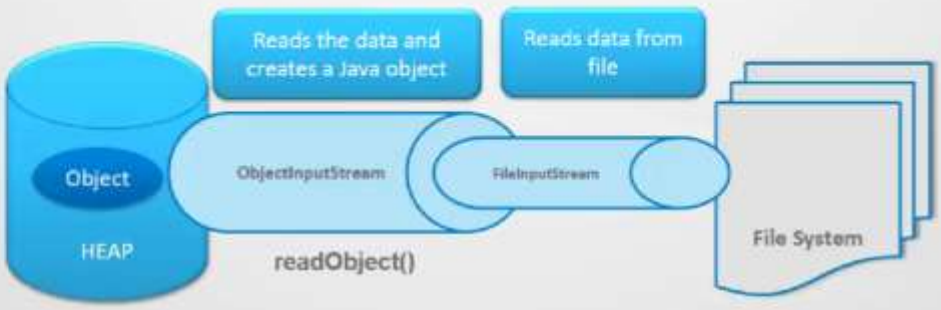
Deserialization

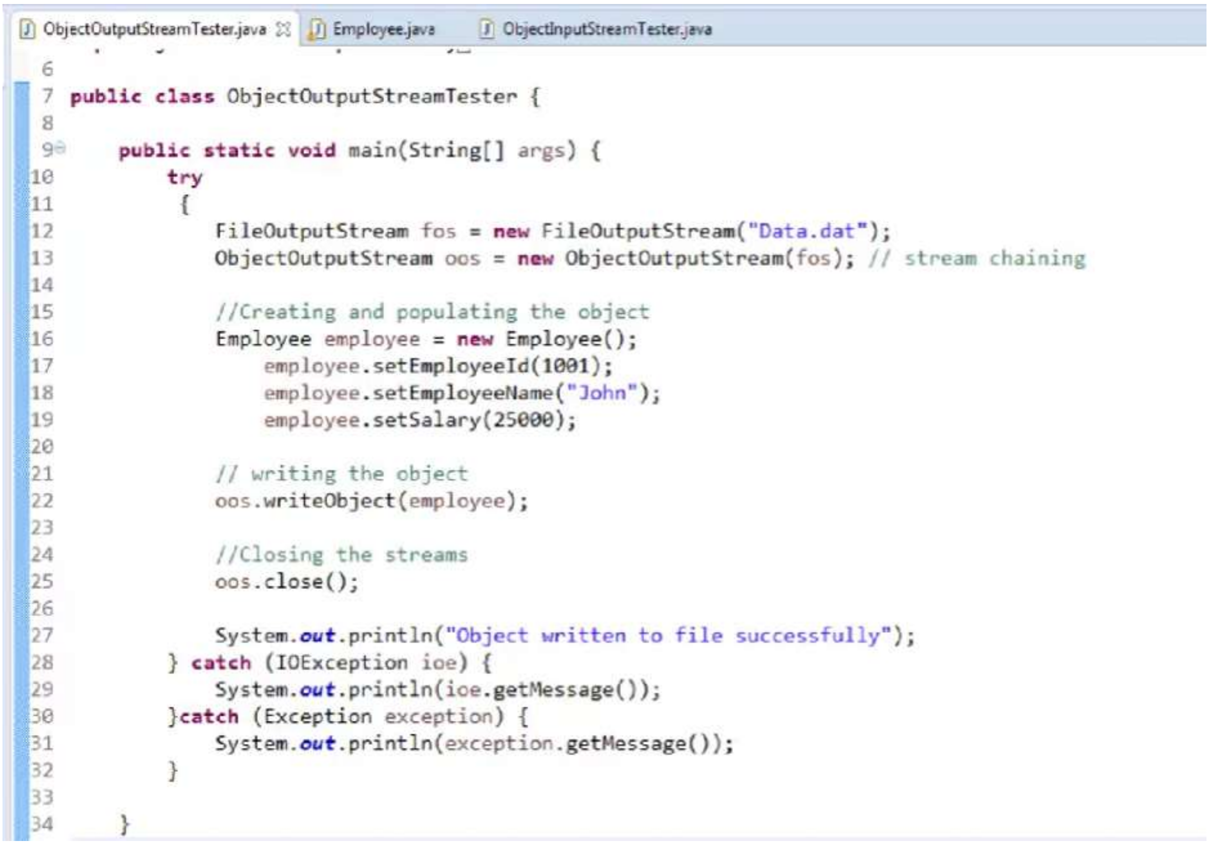
- Deserialization is the process of retrieving an object from the byte streams
- It is achieved using ObjectInputStream



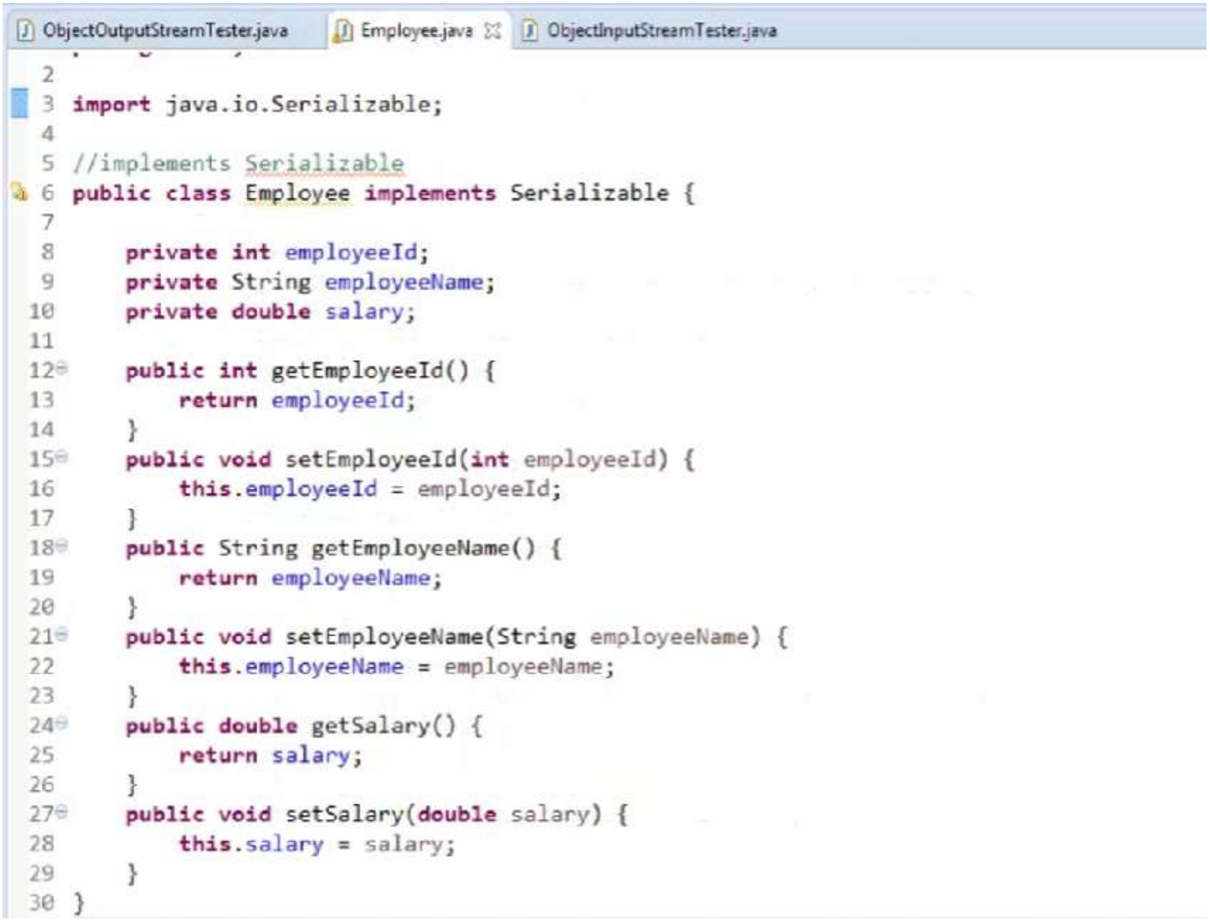
Deserialization

- Deserialization is the process of retrieving an object from the byte streams
- It is achieved using ObjectInputStream

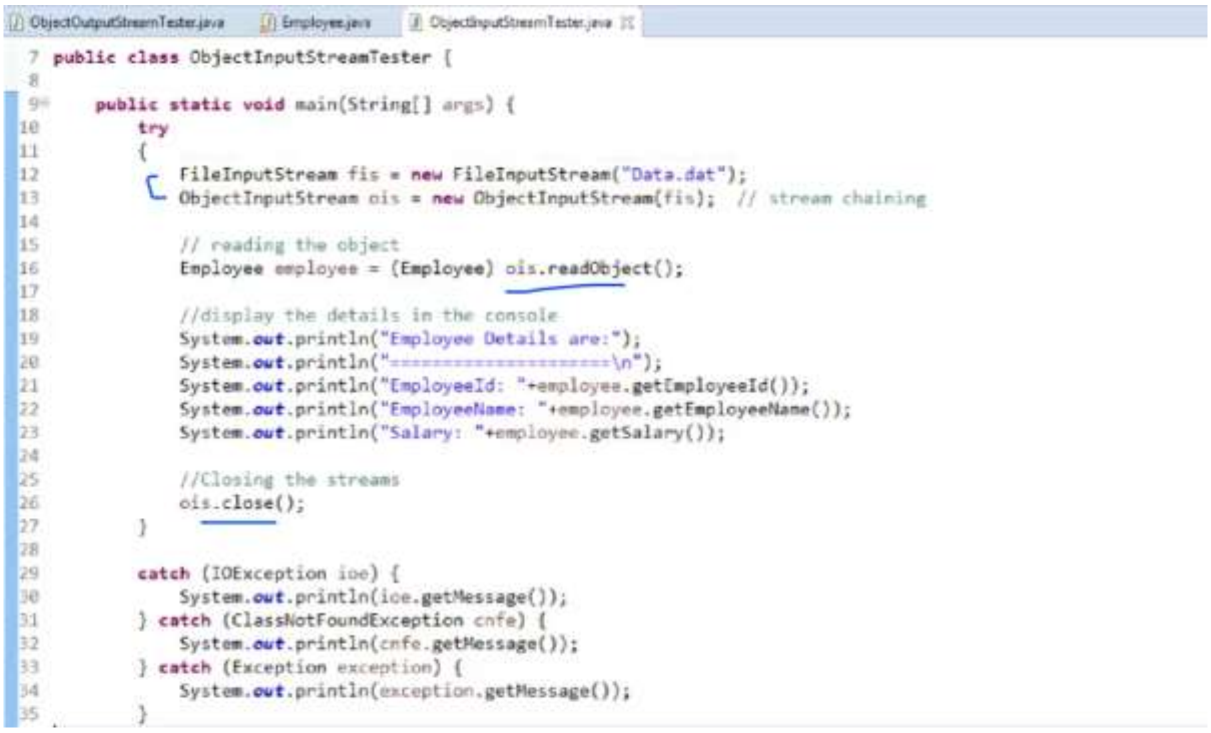




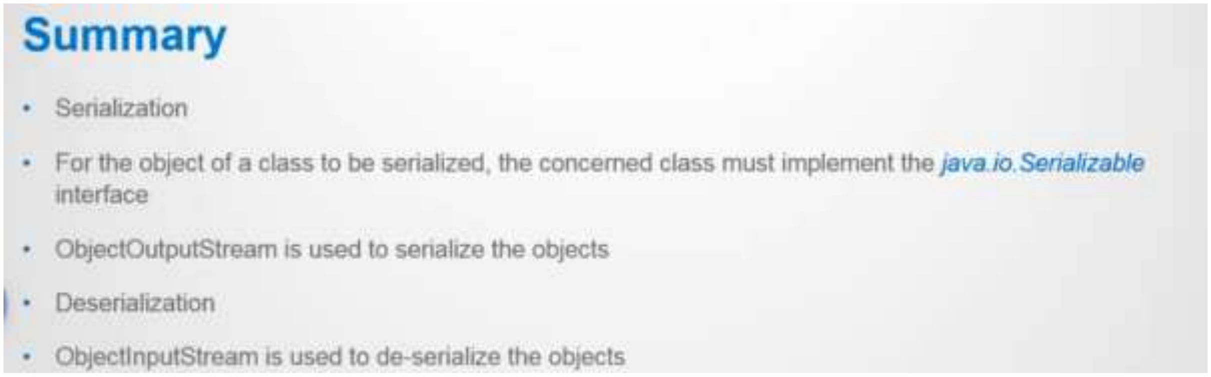
```java

public class ObjectOutputStreamTester {

    public static void main(String[] args) {
        try
         {
            FileOutputStream fos = new FileOutputStream("Data.dat");
            ObjectOutputStream oos = new ObjectOutputStream(fos); // stream chaining

            //Creating and populating the object
            Employee employee = new Employee();
                employee.setEmployeeId(1001);
                employee.setEmployeeName("John");
                employee.setSalary(25000);

            // writing the object
            oos.writeObject(employee);

            //Closing the streams
            oos.close();

            System.out.println("Object written to file successfully");
        } catch (IOException ioe) {
            System.out.println(ioe.getMessage());
        }catch (Exception exception) {
            System.out.println(exception.getMessage());
        }

    }
```



```java

import java.io.Serializable;

//implements Serializable
public class Employee implements Serializable {

    private int employeeId;
    private String employeeName;
    private double salary;

    public int getEmployeeId() {
        return employeeId;
    }
    public void setEmployeeId(int employeeId) {
        this.employeeId = employeeId;
    }
    public String getEmployeeName() {
        return employeeName;
    }
    public void setEmployeeName(String employeeName) {
        this.employeeName = employeeName;
    }
    public double getSalary() {
        return salary;
    }
    public void setSalary(double salary) {
        this.salary = salary;
    }
}
```



```
public class ObjectInputStreamTester {

    public static void main(String[] args) {
        try
        {
            FileInputStream fis = new FileInputStream("Data.dat");
            ObjectInputStream ois = new ObjectInputStream(fis);  // stream chaining

            // reading the object
            Employee employee = (Employee) ois.readObject();

            //display the details in the console
            System.out.println("Employee Details are:");
            System.out.println("=====================\n");
            System.out.println("EmployeeId: "+employee.getEmployeeId());
            System.out.println("EmployeeName: "+employee.getEmployeeName());
            System.out.println("Salary: "+employee.getSalary());

            //Closing the streams
            ois.close();
        }

        catch (IOException ioe) {
            System.out.println(ioe.getMessage());
        } catch (ClassNotFoundException cnfe) {
            System.out.println(cnfe.getMessage());
        } catch (Exception exception) {
            System.out.println(exception.getMessage());
        }
```

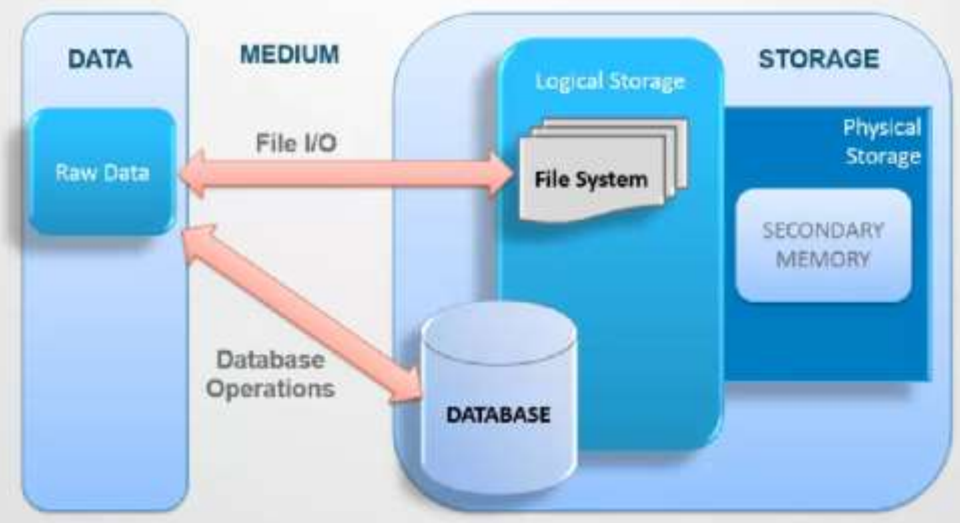


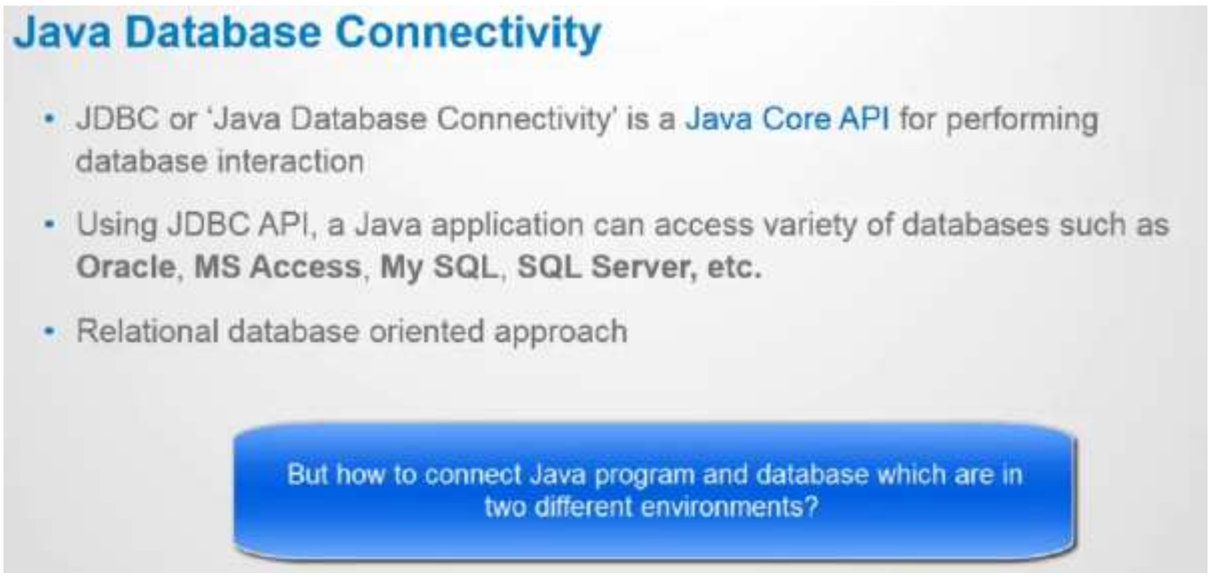
Summary

- Serialization
- For the object of a class to be serialized, the concerned class must implement the *java.io.Serializable* interface
- ObjectOutputStream is used to serialize the objects
- Deserialization
- ObjectInputStream is used to de-serialize the objects

JDBC:

Data Persistence



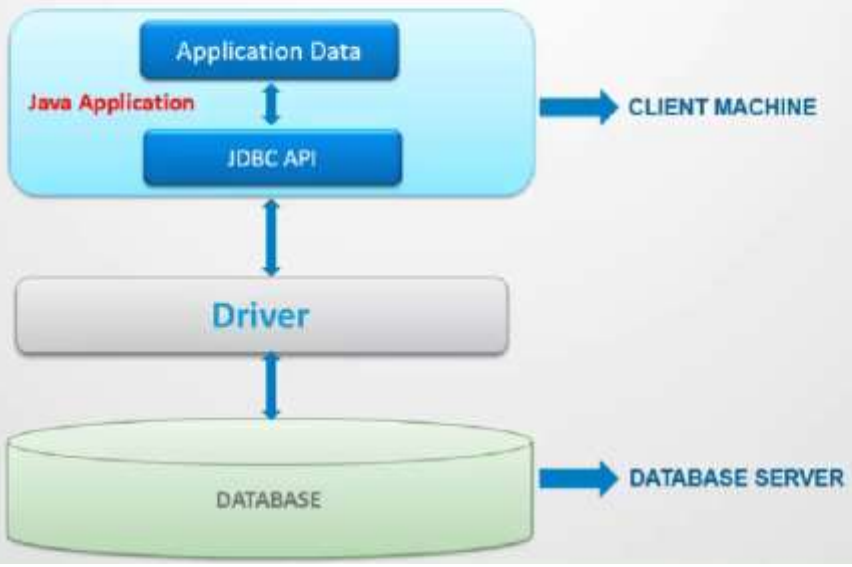


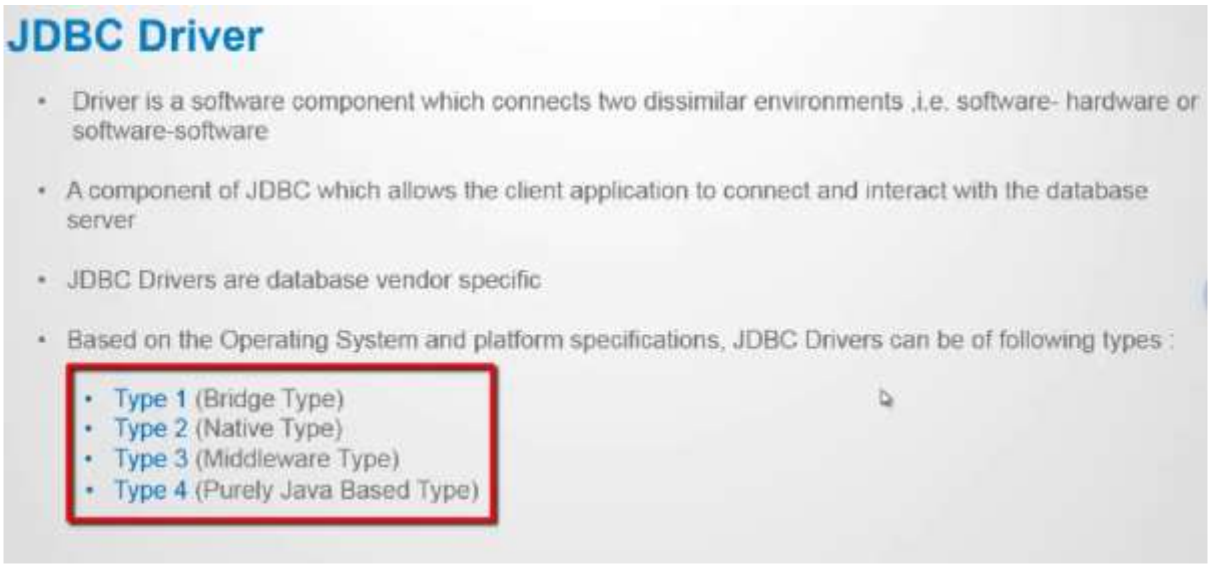
Java Database Connectivity

- JDBC or 'Java Database Connectivity' is a Java Core API for performing database interaction
- Using JDBC API, a Java application can access variety of databases such as **Oracle**, **MS Access**, **My SQL**, **SQL Server, etc.**
- Relational database oriented approach

But how to connect Java program and database which are in two different environments?

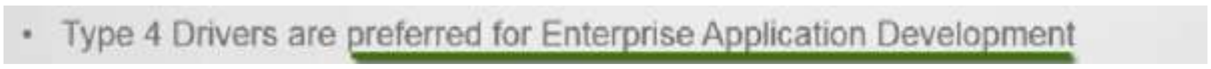
JDBC Overview



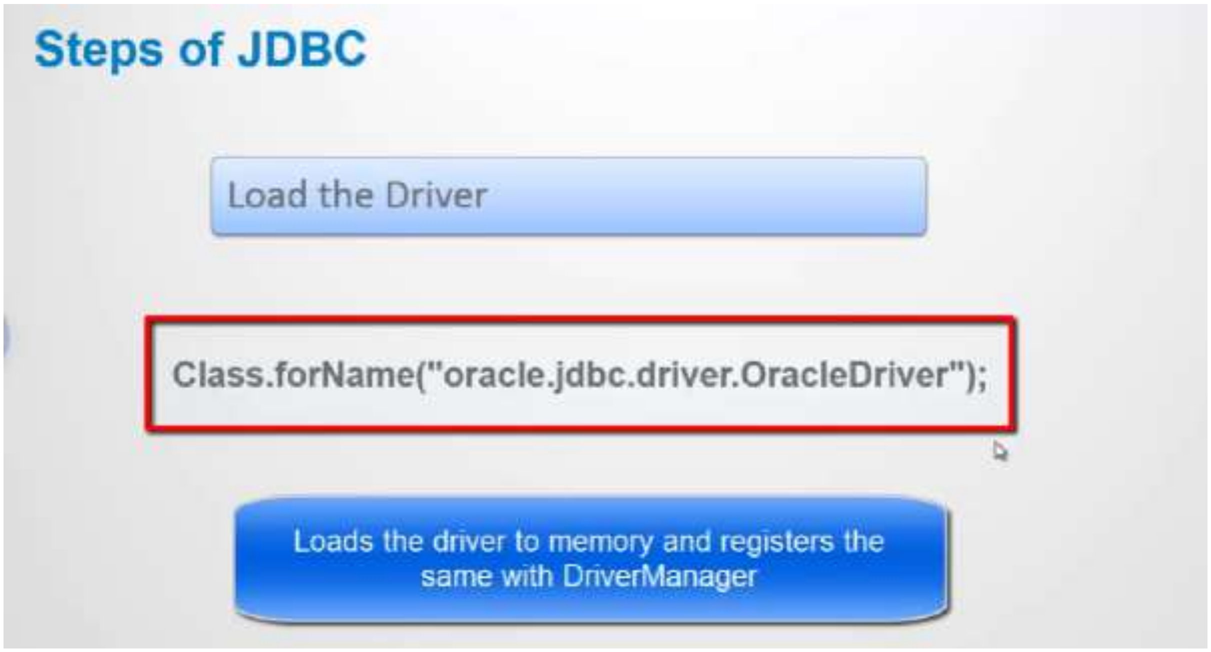


JDBC Driver

- Driver is a software component which connects two dissimilar environments ,i.e. software- hardware or software-software
- A component of JDBC which allows the client application to connect and interact with the database server
- JDBC Drivers are database vendor specific
- Based on the Operating System and platform specifications, JDBC Drivers can be of following types :
  - Type 1 (Bridge Type)
  - Type 2 (Native Type)
  - Type 3 (Middleware Type)
  - Type 4 (Purely Java Based Type)



- Type 4 Drivers are preferred for Enterprise Application Development

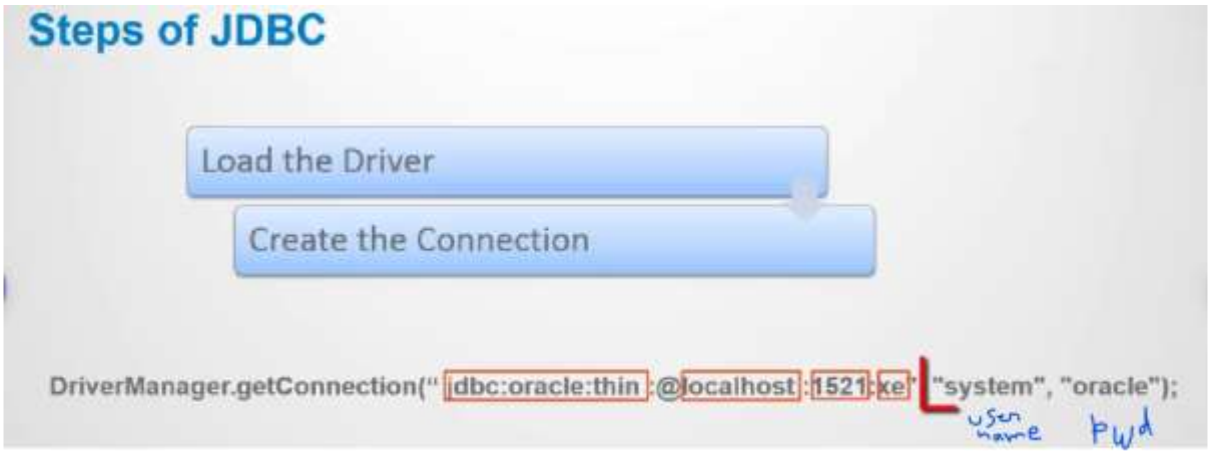


Steps of JDBC

Load the Driver

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

Loads the driver to memory and registers the same with DriverManager



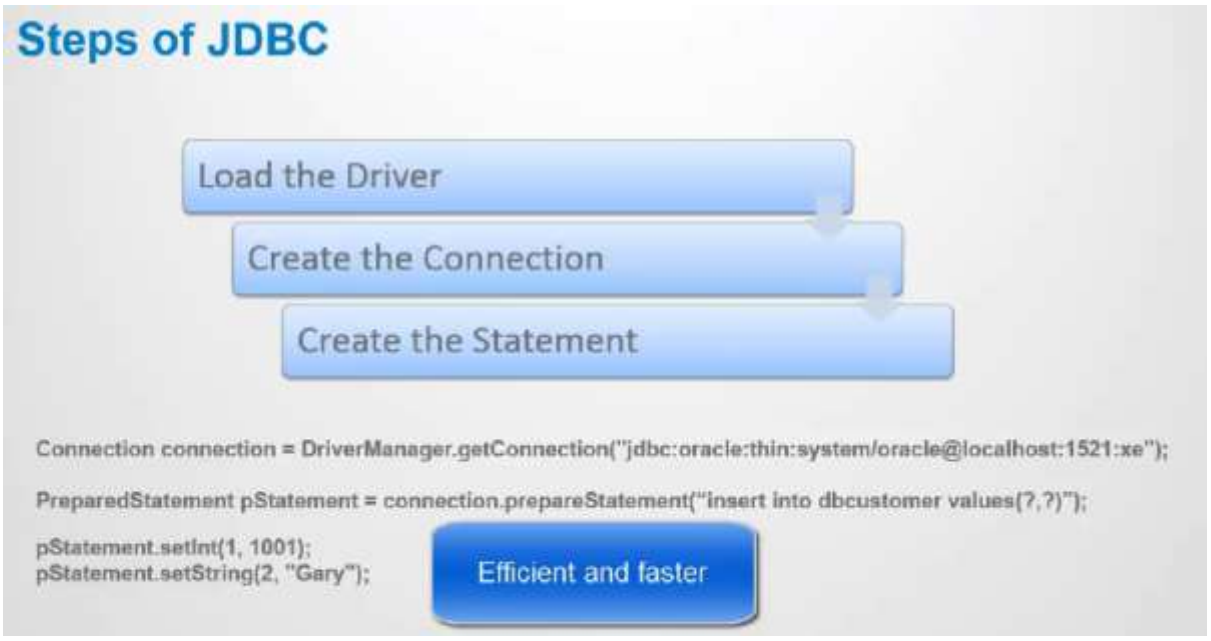
Steps of JDBC

Load the Driver

Create the Connection

```
DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system", "oracle");
```

user name pwd



Steps of JDBC

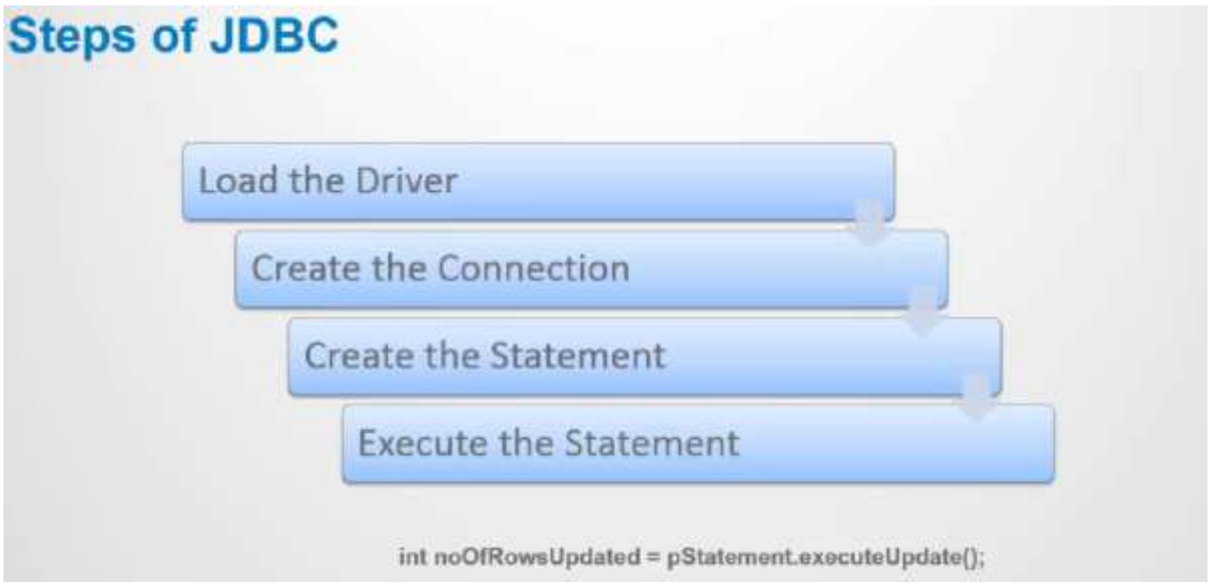
Load the Driver

Create the Connection

Create the Statement

```
Connection connection = DriverManager.getConnection("jdbc:oracle:thin:system/oracle@localhost:1521:xe");

PreparedStatement pStatement = connection.prepareStatement("insert into dbcustomer values(?,?)");

pStatement.setInt(1, 1001);
pStatement.setString(2, "Gary");
```

Efficient and faster



Steps of JDBC

Load the Driver

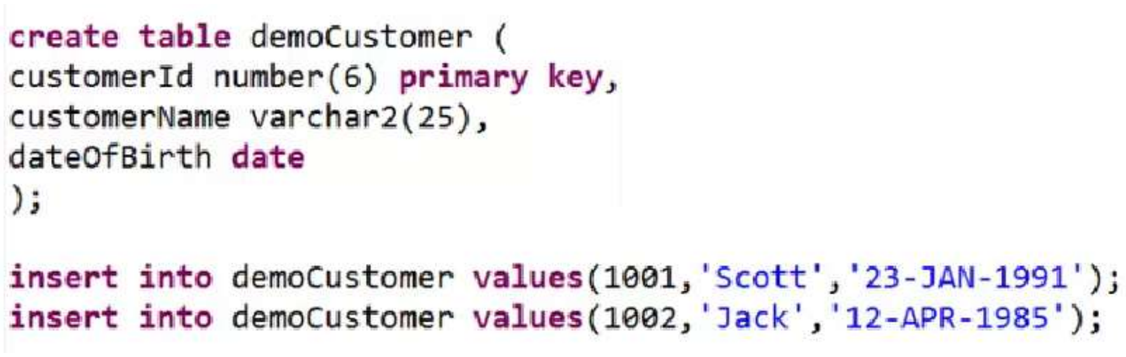
Create the Connection

Create the Statement

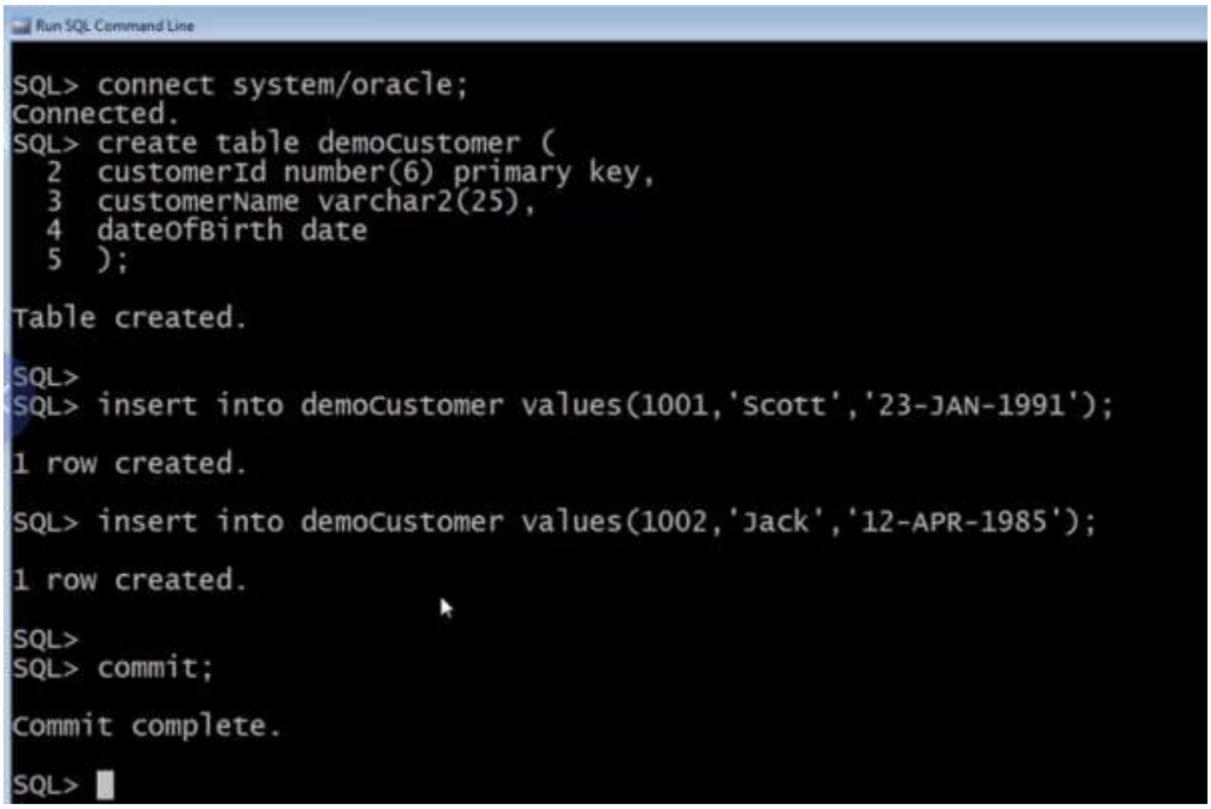
Execute the Statement

```
int noOfRowsUpdated = pStatement.executeUpdate();
```

To see more clearly lets create a table

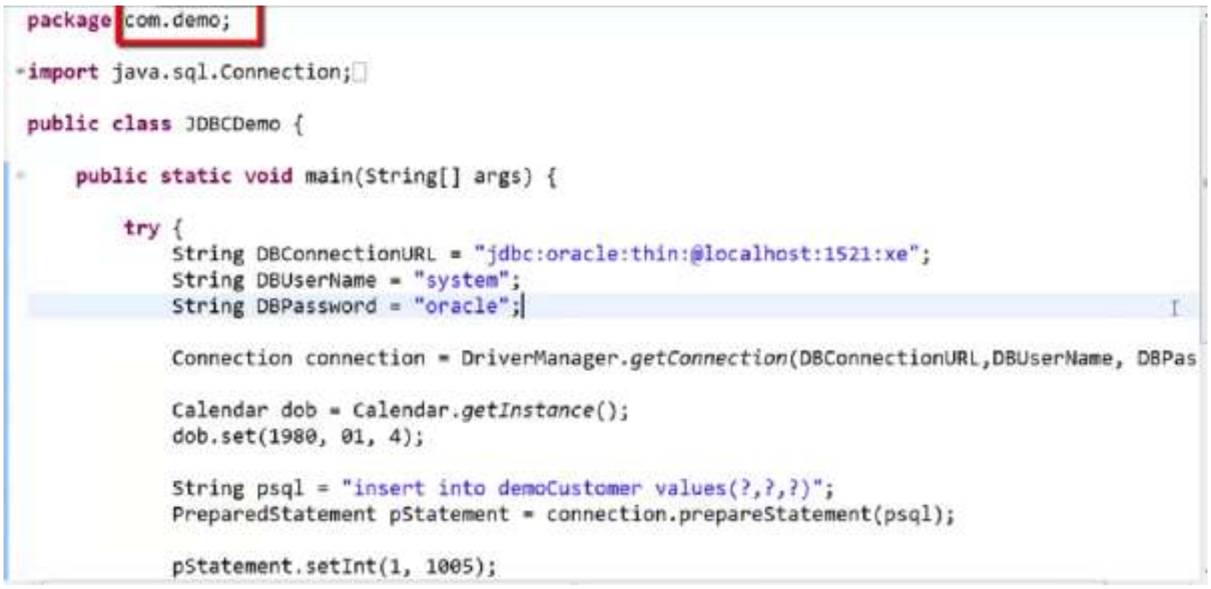


```
create table demoCustomer (
customerId number(6) primary key,
customerName varchar2(25),
dateOfBirth date
);

insert into demoCustomer values(1001,'Scott','23-JAN-1991');
insert into demoCustomer values(1002,'Jack','12-APR-1985');
```

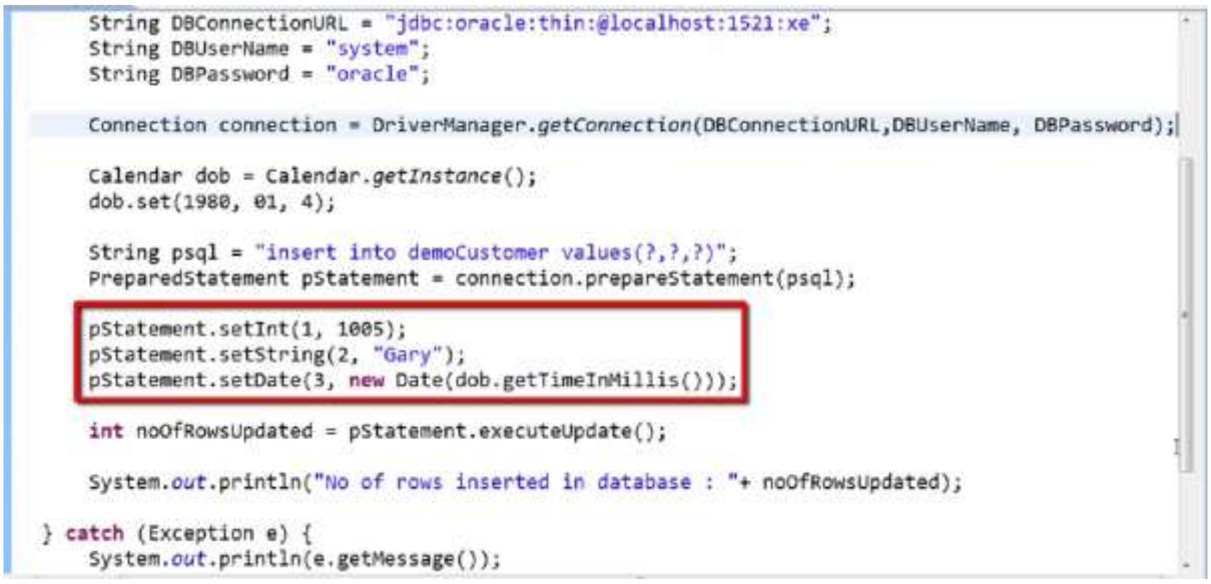


```
SQL> connect system/oracle;
Connected.
SQL> create table demoCustomer (
  2  customerId number(6) primary key,
  3  customerName varchar2(25),
  4  dateOfBirth date
  5  );

Table created.

SQL>
SQL> insert into demoCustomer values(1001,'Scott','23-JAN-1991');

1 row created.

SQL> insert into demoCustomer values(1002,'Jack','12-APR-1985');

1 row created.

SQL>
SQL> commit;

Commit complete.

SQL>
```

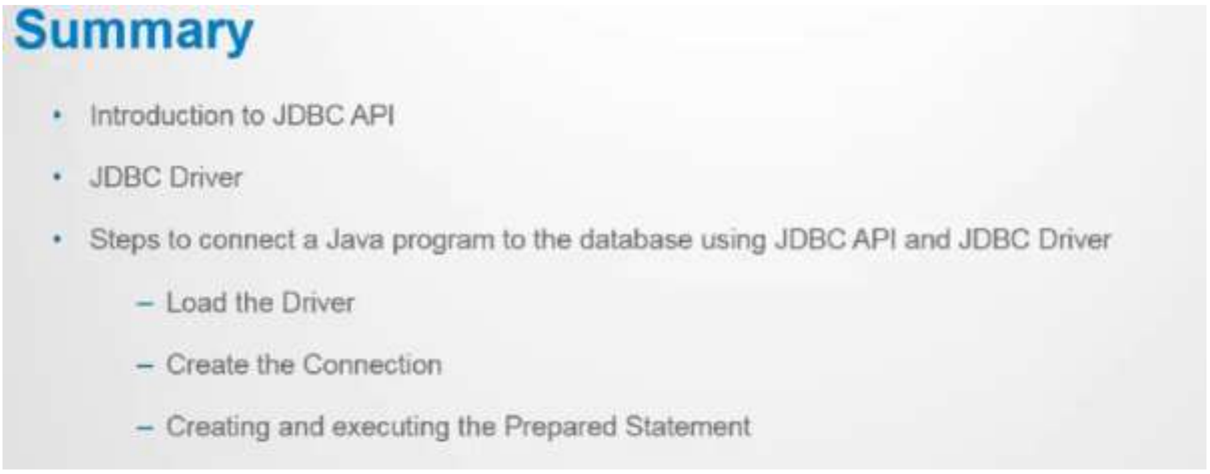
Odbc.jar file should be added to java build path



```java
package com.demo;

import java.sql.Connection;

public class JDBCDemo {

    public static void main(String[] args) {

        try {
            String DBConnectionURL = "jdbc:oracle:thin:@localhost:1521:xe";
            String DBUserName = "system";
            String DBPassword = "oracle";

            Connection connection = DriverManager.getConnection(DBConnectionURL,DBUserName, DBPas

            Calendar dob = Calendar.getInstance();
            dob.set(1980, 01, 4);

            String psql = "insert into demoCustomer values(?,?,?)";
            PreparedStatement pStatement = connection.prepareStatement(psql);

            pStatement.setInt(1, 1005);
```



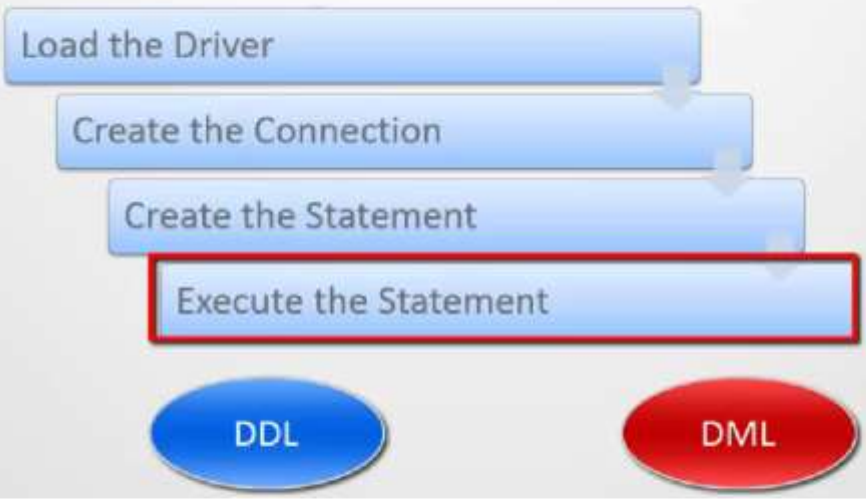
```
    String DBConnectionURL = "jdbc:oracle:thin:@localhost:1521:xe";
    String DBUserName = "system";
    String DBPassword = "oracle";

    Connection connection = DriverManager.getConnection(DBConnectionURL,DBUserName, DBPassword);

    Calendar dob = Calendar.getInstance();
    dob.set(1980, 01, 4);

    String psql = "insert into demoCustomer values(?,?,?)";
    PreparedStatement pStatement = connection.prepareStatement(psql);

    pStatement.setInt(1, 1005);
    pStatement.setString(2, "Gary");
    pStatement.setDate(3, new Date(dob.getTimeInMillis()));

    int noOfRowsUpdated = pStatement.executeUpdate();

    System.out.println("No of rows inserted in database : "+ noOfRowsUpdated);

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

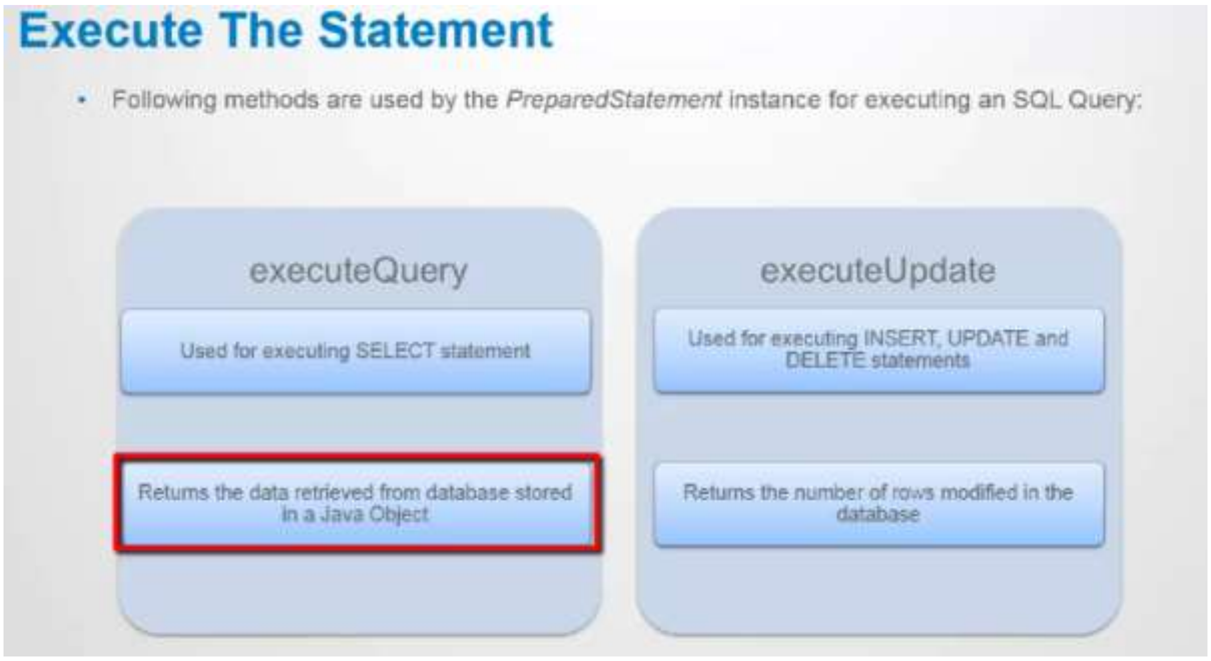


Summary

- Introduction to JDBC API
- JDBC Driver
- Steps to connect a Java program to the database using JDBC API and JDBC Driver
  - Load the Driver
  - Create the Connection
  - Creating and executing the Prepared Statement

Steps of JDBC





Execute The Statement

- Following methods are used by the *PreparedStatement* instance for executing an SQL Query:

executeQuery

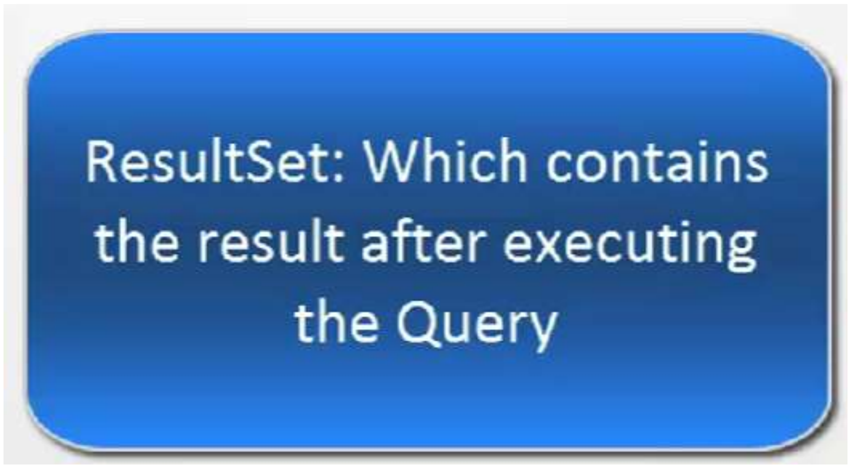
Used for executing SELECT statement

Returns the data retrieved from database stored in a Java Object

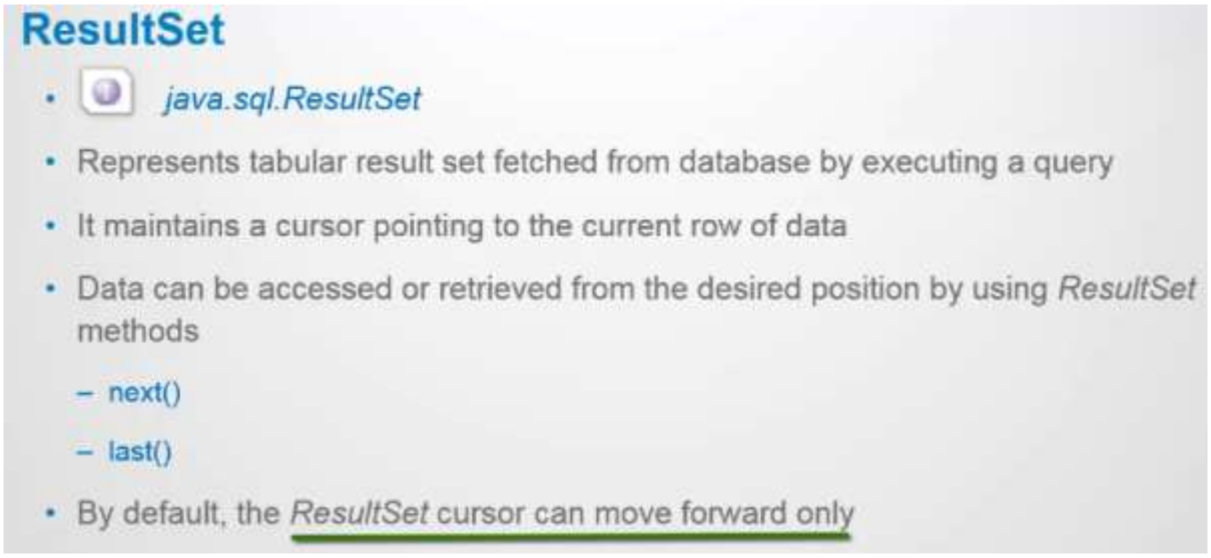
executeUpdate

Used for executing INSERT, UPDATE and DELETE statements

Returns the number of rows modified in the database



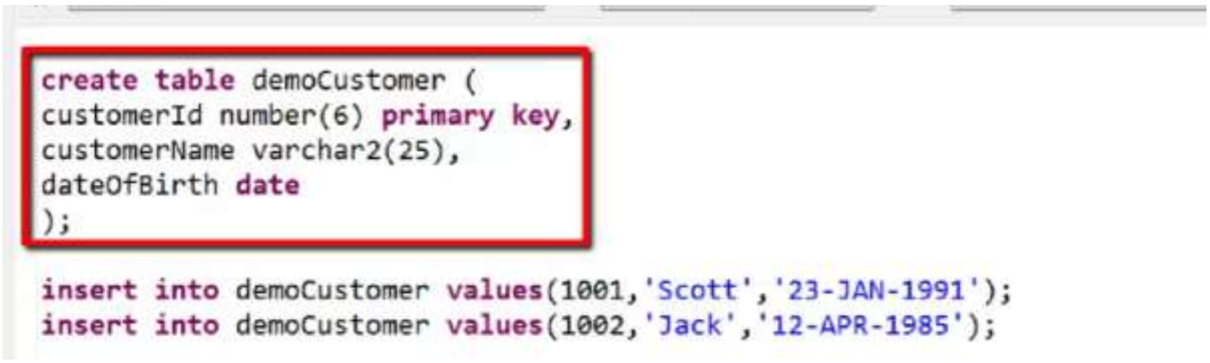
ResultSet: Which contains the result after executing the Query



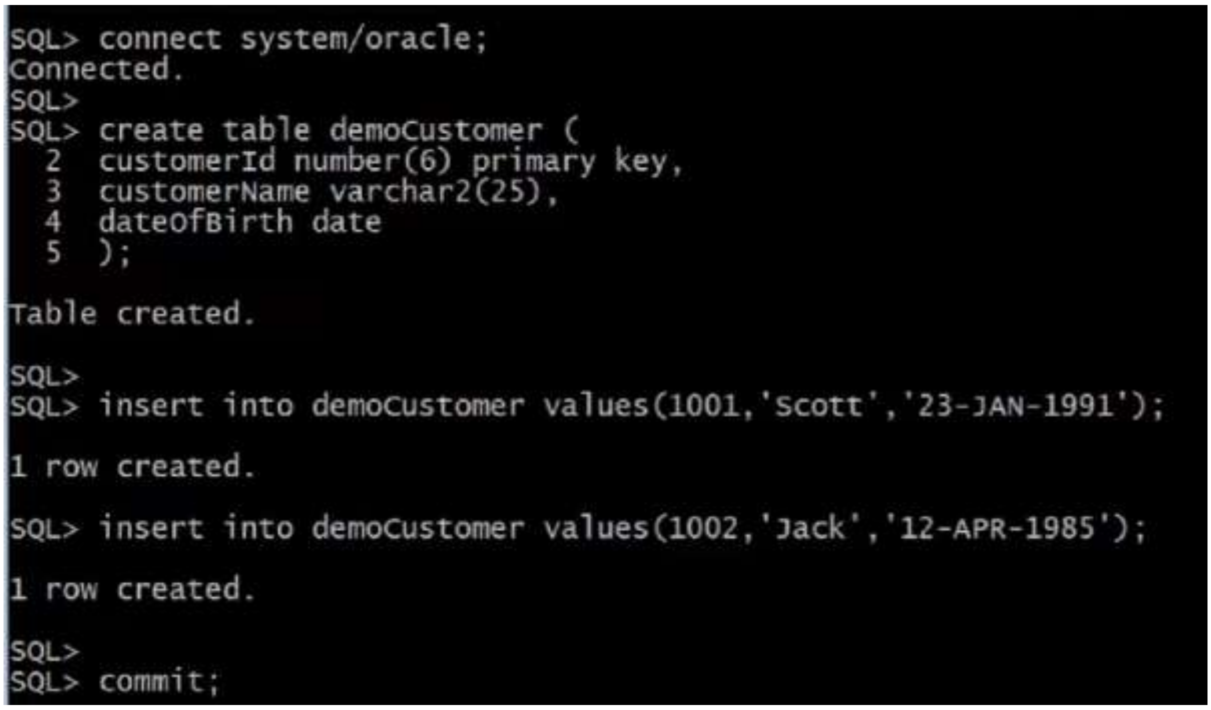
ResultSet

- *java.sql.ResultSet*
- Represents tabular result set fetched from database by executing a query
- It maintains a cursor pointing to the current row of data
- Data can be accessed or retrieved from the desired position by using *ResultSet* methods
  - next()
  - last()
- By default, the *ResultSet* cursor can move forward only

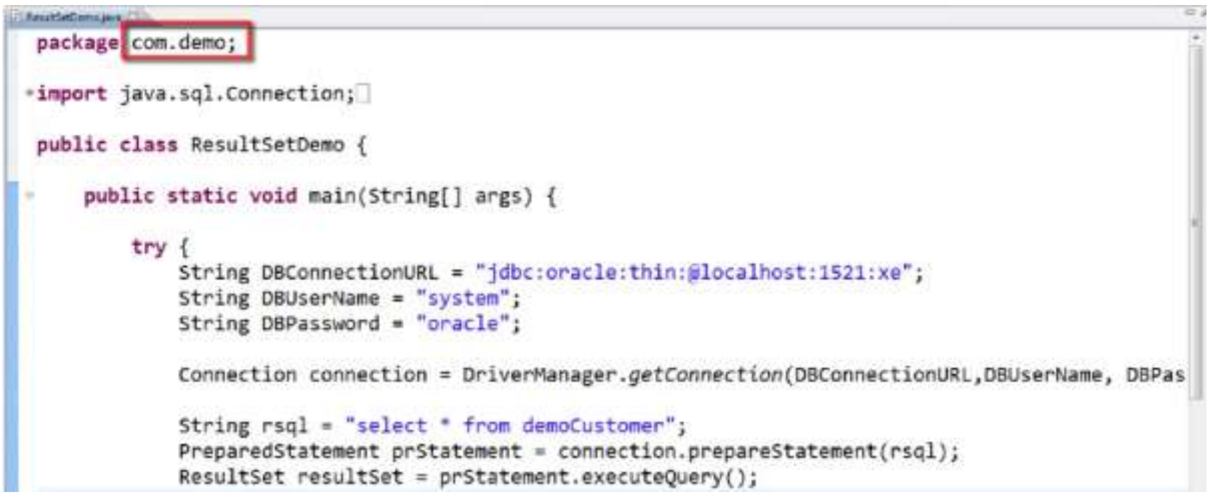
Lets see demo



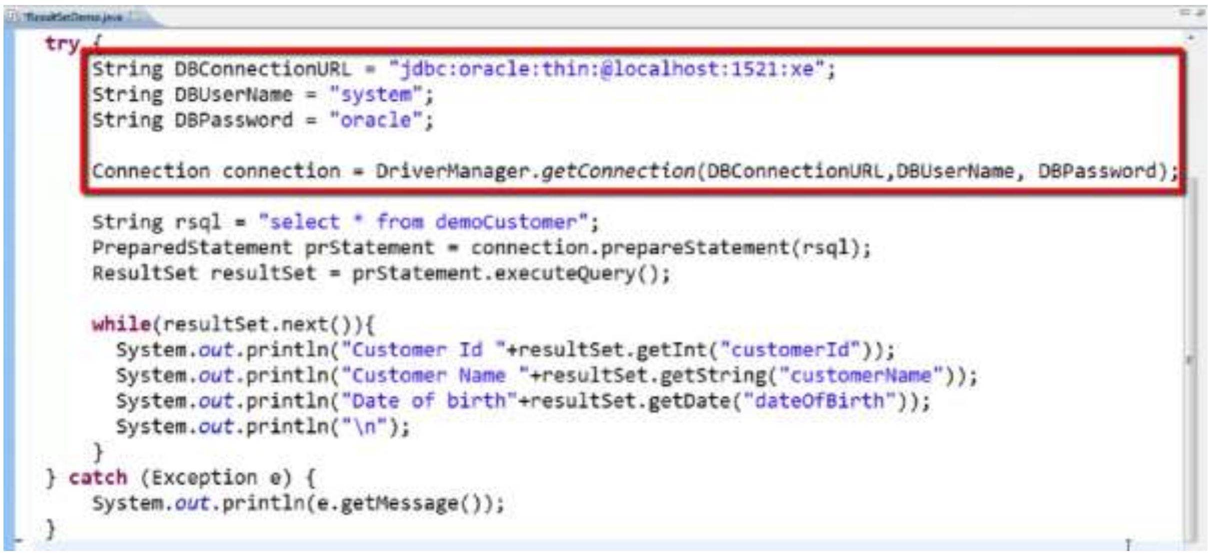
```
create table demoCustomer (
customerId number(6) primary key,
customerName varchar2(25),
dateOfBirth date
);

insert into demoCustomer values(1001,'Scott','23-JAN-1991');
insert into demoCustomer values(1002,'Jack','12-APR-1985');
```



```
SQL> connect system/oracle;
Connected.
SQL>
SQL> create table demoCustomer (
  2  customerId number(6) primary key,
  3  customerName varchar2(25),
  4  dateOfBirth date
  5  );

Table created.

SQL>
SQL> insert into demoCustomer values(1001,'Scott','23-JAN-1991');

1 row created.

SQL> insert into demoCustomer values(1002,'Jack','12-APR-1985');

1 row created.

SQL>
SQL> commit;
```



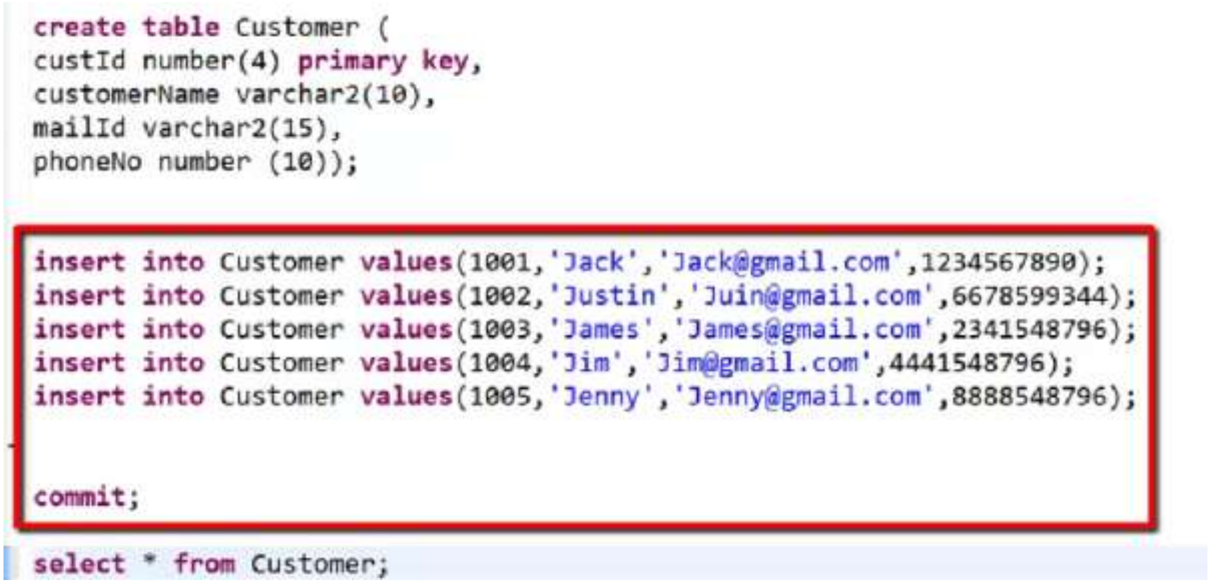
```
package com.demo;

import java.sql.Connection;

public class ResultSetDemo {

    public static void main(String[] args) {

        try {
            String DBConnectionURL = "jdbc:oracle:thin:@localhost:1521:xe";
            String DBUserName = "system";
            String DBPassword = "oracle";

            Connection connection = DriverManager.getConnection(DBConnectionURL,DBUserName, DBPas

            String rsql = "select * from demoCustomer";
            PreparedStatement prStatement = connection.prepareStatement(rsql);
            ResultSet resultSet = prStatement.executeQuery();
```



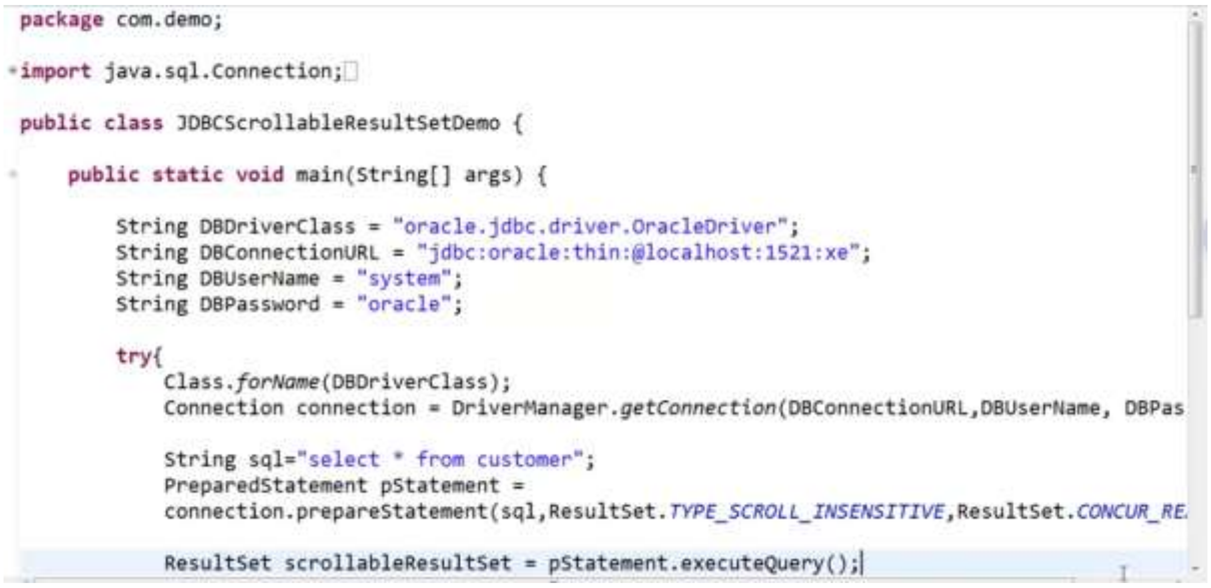
```
try {
    String DBConnectionURL = "jdbc:oracle:thin:@localhost:1521:xe";
    String DBUserName = "system";
    String DBPassword = "oracle";

    Connection connection = DriverManager.getConnection(DBConnectionURL,DBUserName, DBPassword);

    String rsql = "select * from demoCustomer";
    PreparedStatement prStatement = connection.prepareStatement(rsql);
    ResultSet resultSet = prStatement.executeQuery();

    while(resultSet.next()){
      System.out.println("Customer Id "+resultSet.getInt("customerId"));
      System.out.println("Customer Name "+resultSet.getString("customerName"));
      System.out.println("Date of birth"+resultSet.getDate("dateOfBirth"));
      System.out.println("\n");
    }
} catch (Exception e) {
    System.out.println(e.getMessage());
}
```

On each invocation of next() method the cursor present inside the resultset will point to the next row and returns true

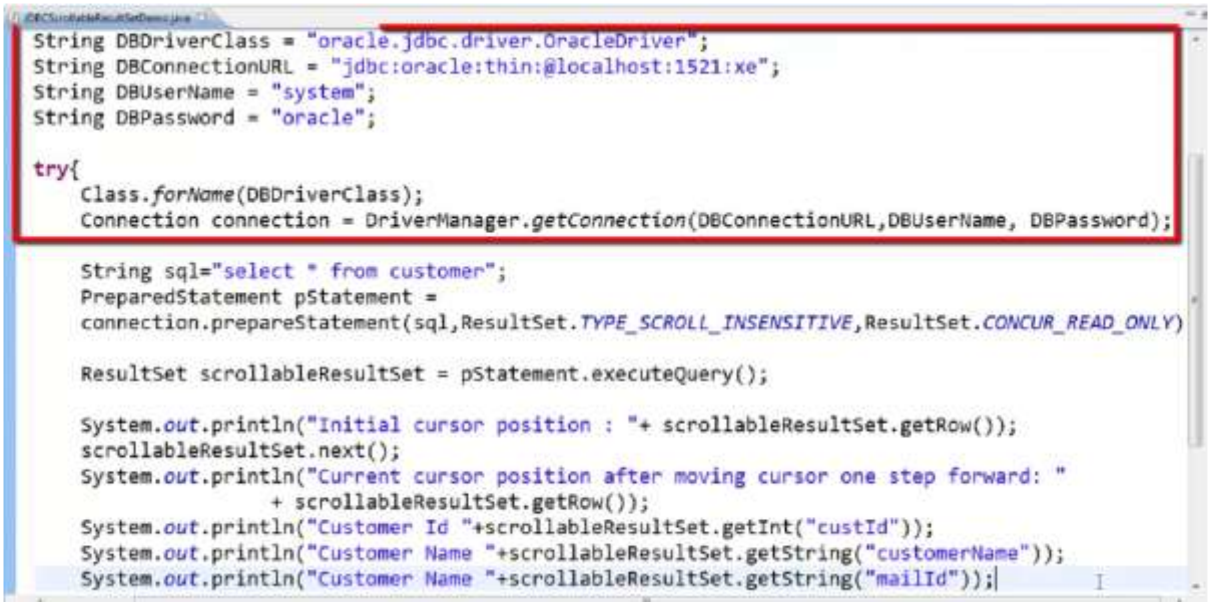
If there are no more rows left to be iterated this next() method will return false



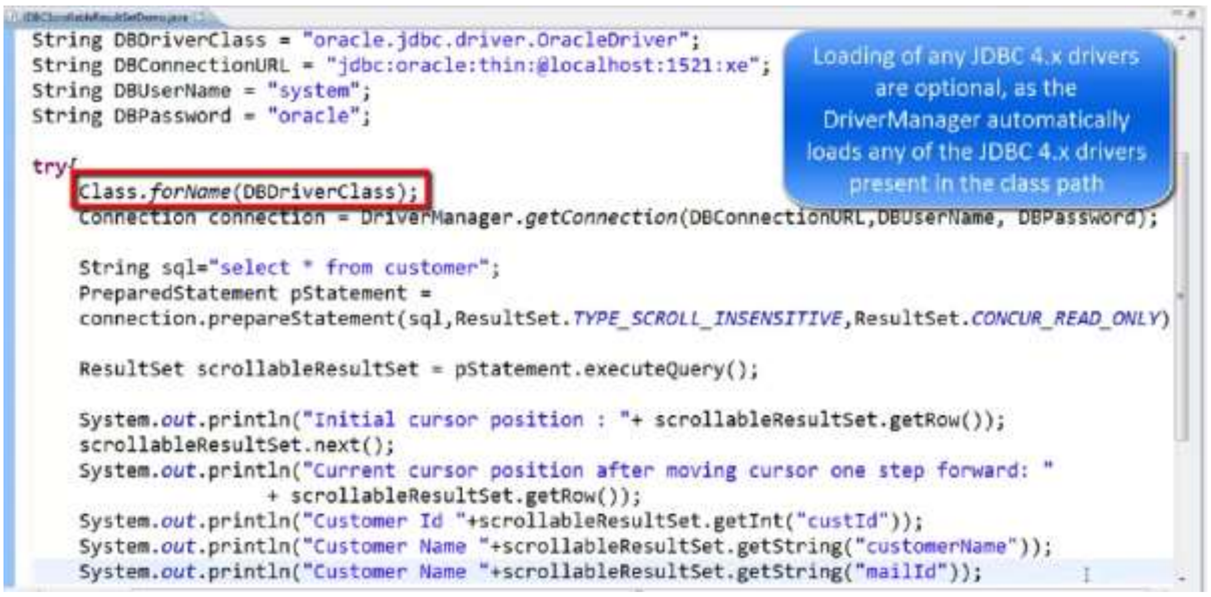
```
create table Customer (
custId number(4) primary key,
customerName varchar2(10),
mailId varchar2(15),
phoneNo number (10));

insert into Customer values(1001,'Jack','Jack@gmail.com',1234567890);
insert into Customer values(1002,'Justin','Juin@gmail.com',6678599344);
insert into Customer values(1003,'James','James@gmail.com',2341548796);
insert into Customer values(1004,'Jim','Jim@gmail.com',4441548796);
insert into Customer values(1005,'Jenny','Jenny@gmail.com',8888548796);


commit;

select * from Customer;
```



```
package com.demo;

import java.sql.Connection;

public class JDBCScrollableResultSetDemo {

    public static void main(String[] args) {

        String DBDriverClass = "oracle.jdbc.driver.OracleDriver";
        String DBConnectionURL = "jdbc:oracle:thin:@localhost:1521:xe";
        String DBUserName = "system";
        String DBPassword = "oracle";

        try{
            Class.forName(DBDriverClass);
            Connection connection = DriverManager.getConnection(DBConnectionURL,DBUserName, DBPas

            String sql="select * from customer";
            PreparedStatement pStatement =
            connection.prepareStatement(sql,ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_RE

            ResultSet scrollableResultSet = pStatement.executeQuery();
```

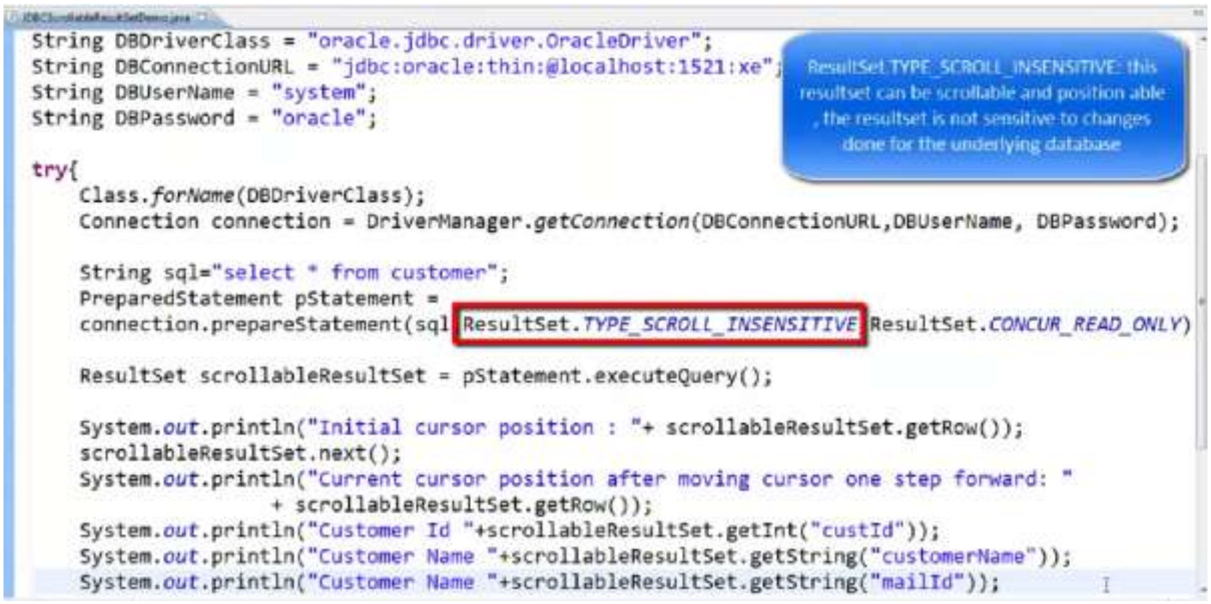


```
String DBDriverClass = "oracle.jdbc.driver.OracleDriver";
String DBConnectionURL = "jdbc:oracle:thin:@localhost:1521:xe";
String DBUserName = "system";
String DBPassword = "oracle";

try{
    Class.forName(DBDriverClass);
    Connection connection = DriverManager.getConnection(DBConnectionURL,DBUserName, DBPassword);

    String sql="select * from customer";
    PreparedStatement pStatement =
    connection.prepareStatement(sql,ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_READ_ONLY)

    ResultSet scrollableResultSet = pStatement.executeQuery();

    System.out.println("Initial cursor position : "+ scrollableResultSet.getRow());
    scrollableResultSet.next();
    System.out.println("Current cursor position after moving cursor one step forward: "
                + scrollableResultSet.getRow());
    System.out.println("Customer Id "+scrollableResultSet.getInt("custId"));
    System.out.println("Customer Name "+scrollableResultSet.getString("customerName"));
    System.out.println("Customer Name "+scrollableResultSet.getString("mailId"));
```



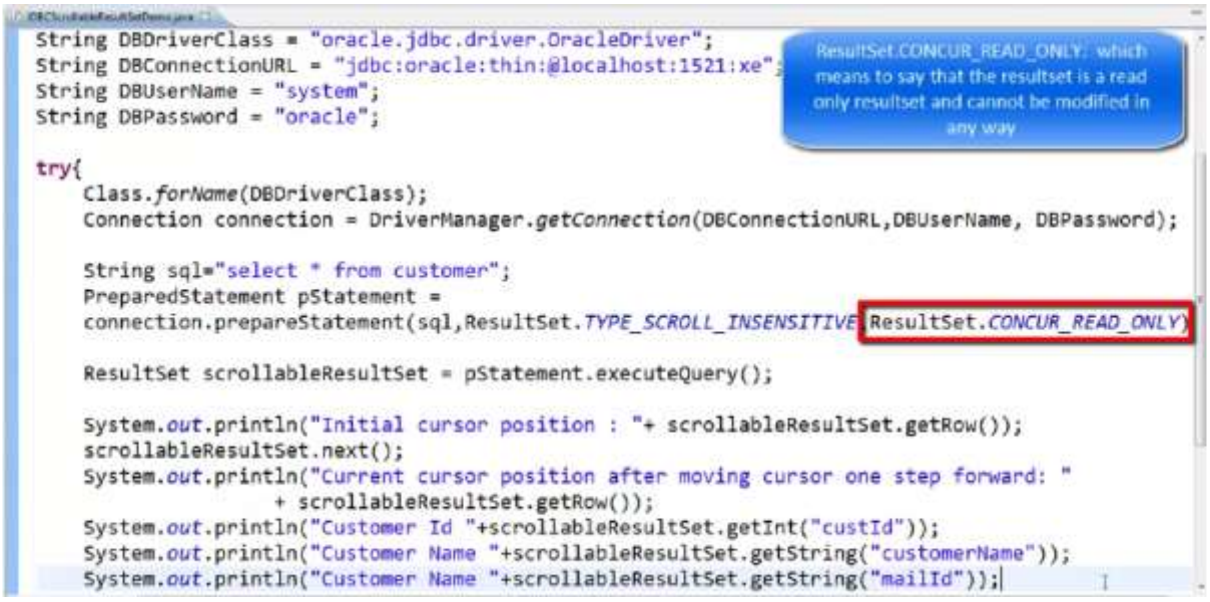
```
String DBDriverClass = "oracle.jdbc.driver.OracleDriver";
String DBConnectionURL = "jdbc:oracle:thin:@localhost:1521:xe";
String DBUserName = "system";
String DBPassword = "oracle";

try{
    Class.forName(DBDriverClass);
    Connection connection = DriverManager.getConnection(DBConnectionURL,DBUserName, DBPassword);

    String sql="select * from customer";
    PreparedStatement pStatement =
    connection.prepareStatement(sql,ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_READ_ONLY)

    ResultSet scrollableResultSet = pStatement.executeQuery();

    System.out.println("Initial cursor position : "+ scrollableResultSet.getRow());
    scrollableResultSet.next();
    System.out.println("Current cursor position after moving cursor one step forward: "
                + scrollableResultSet.getRow());
    System.out.println("Customer Id "+scrollableResultSet.getInt("custId"));
    System.out.println("Customer Name "+scrollableResultSet.getString("customerName"));
    System.out.println("Customer Name "+scrollableResultSet.getString("mailId"));
```

Loading of any JDBC 4.x drivers are optional, as the DriverManager automatically loads any of the JDBC 4.x drivers present in the class path



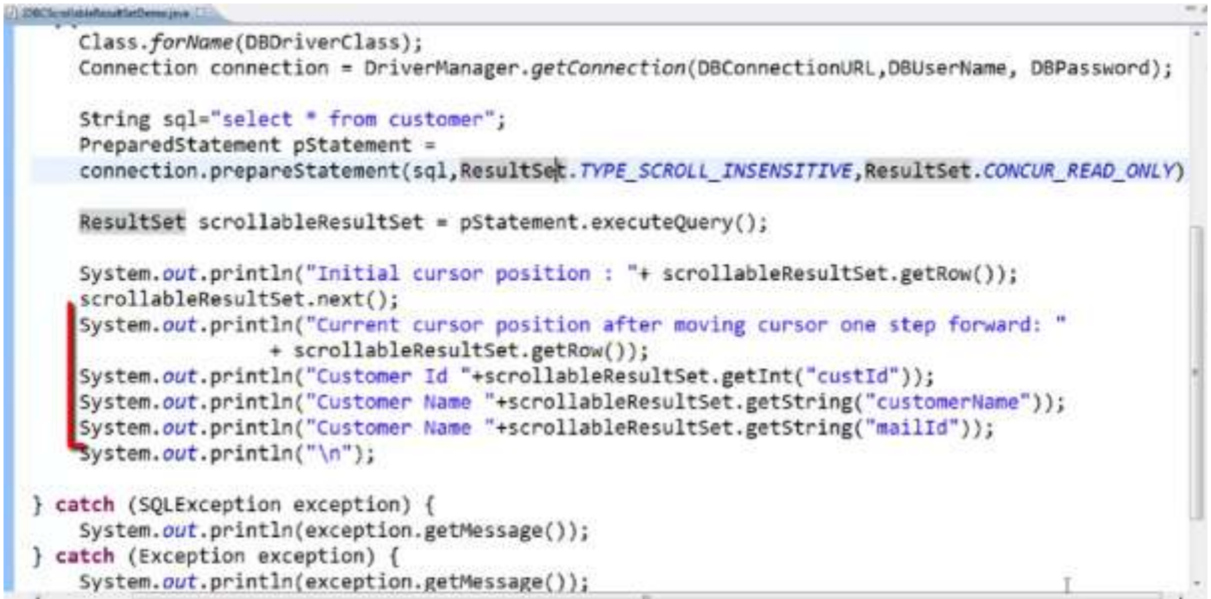
```
String DBDriverClass = "oracle.jdbc.driver.OracleDriver";
String DBConnectionURL = "jdbc:oracle:thin:@localhost:1521:xe";
String DBUserName = "system";
String DBPassword = "oracle";

try{
    Class.forName(DBDriverClass);
    Connection connection = DriverManager.getConnection(DBConnectionURL,DBUserName, DBPassword);

    String sql="select * from customer";
    PreparedStatement pStatement =
    connection.prepareStatement(sql,ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_READ_ONLY);

    ResultSet scrollableResultSet = pStatement.executeQuery();

    System.out.println("Initial cursor position : "+ scrollableResultSet.getRow());
    scrollableResultSet.next();
    System.out.println("Current cursor position after moving cursor one step forward: "
                  + scrollableResultSet.getRow());
    System.out.println("Customer Id "+scrollableResultSet.getInt("custId"));
    System.out.println("Customer Name "+scrollableResultSet.getString("customerName"));
    System.out.println("Customer Name "+scrollableResultSet.getString("mailId"));
```

ResultSet.TYPE_SCROLL_INSENSITIVE: this resultset can be scrollable and position able , the resultset is not sensitive to changes done for the underlying database

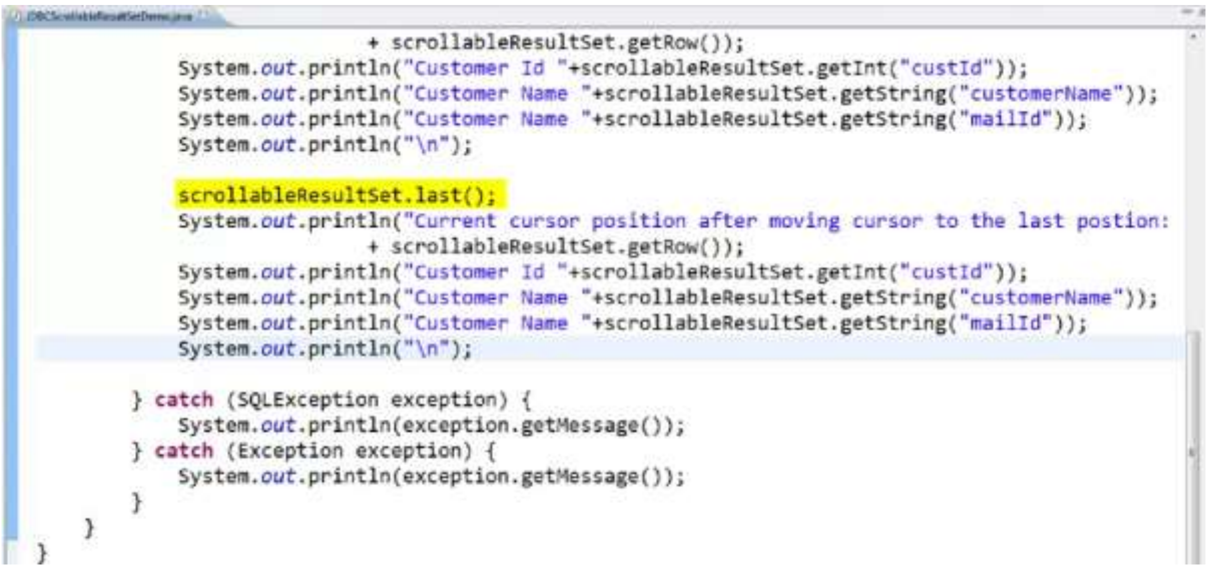


```
String DBDriverClass = "oracle.jdbc.driver.OracleDriver";
String DBConnectionURL = "jdbc:oracle:thin:@localhost:1521:xe";
String DBUserName = "system";
String DBPassword = "oracle";

try{
    Class.forName(DBDriverClass);
    Connection connection = DriverManager.getConnection(DBConnectionURL,DBUserName, DBPassword);

    String sql="select * from customer";
    PreparedStatement pStatement =
    connection.prepareStatement(sql,ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_READ_ONLY);

    ResultSet scrollableResultSet = pStatement.executeQuery();

    System.out.println("Initial cursor position : "+ scrollableResultSet.getRow());
    scrollableResultSet.next();
    System.out.println("Current cursor position after moving cursor one step forward: "
                  + scrollableResultSet.getRow());
    System.out.println("Customer Id "+scrollableResultSet.getInt("custId"));
    System.out.println("Customer Name "+scrollableResultSet.getString("customerName"));
    System.out.println("Customer Name "+scrollableResultSet.getString("mailId"));
```

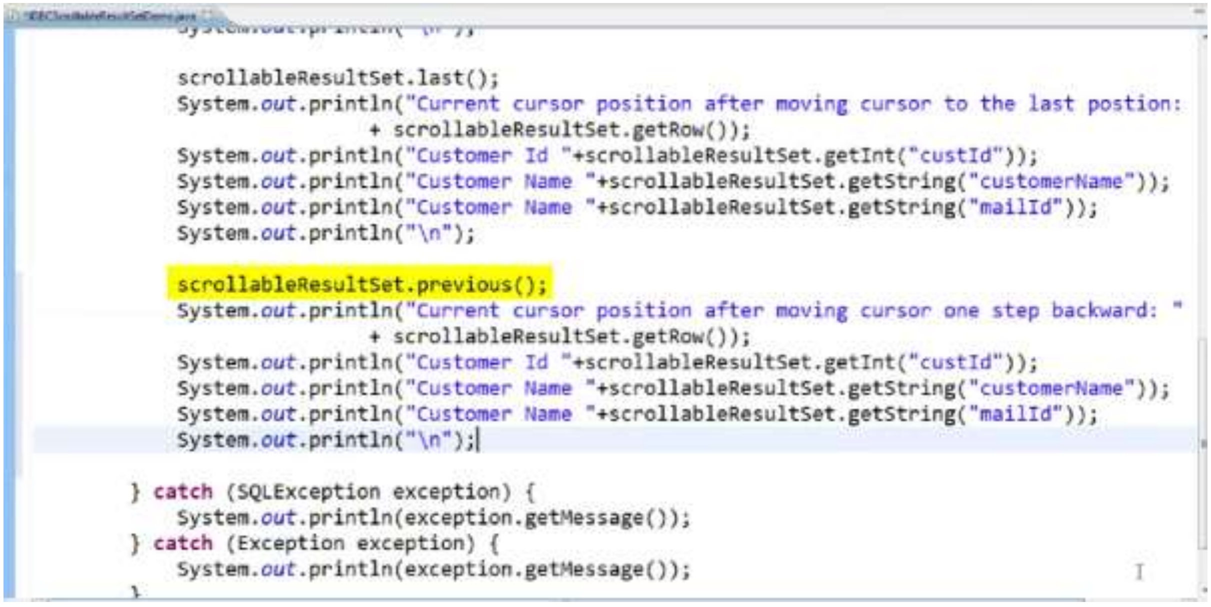
ResultSet.CONCUR_READ_ONLY: which means to say that the resultset is a read only resultset and cannot be modified in any way



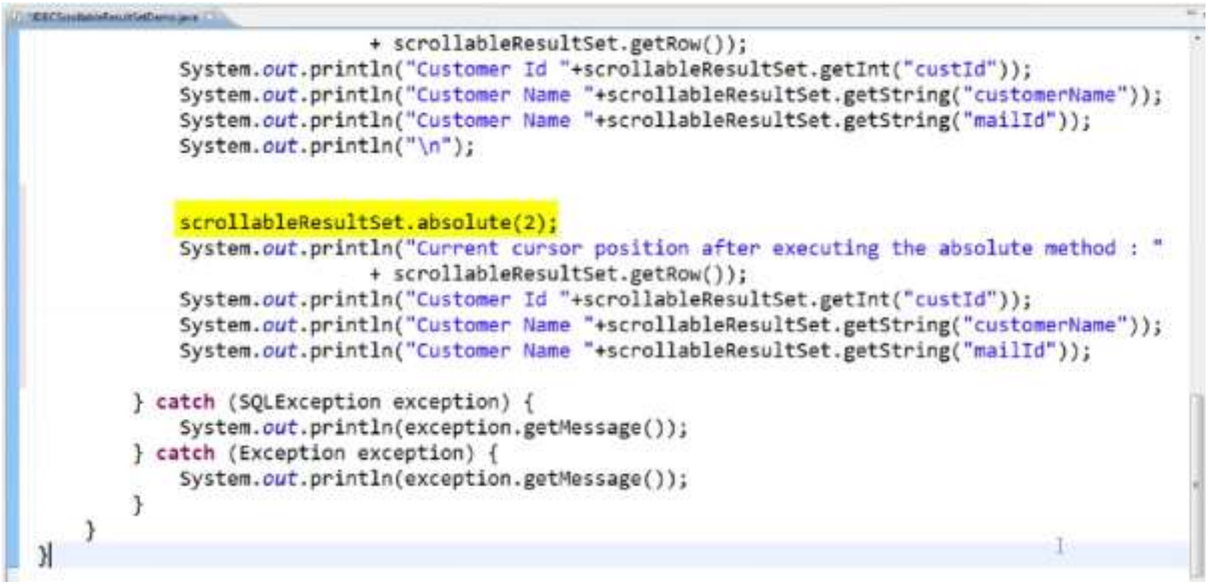
```
    Class.forName(DBDriverClass);
    Connection connection = DriverManager.getConnection(DBConnectionURL,DBUserName, DBPassword);

    String sql="select * from customer";
    PreparedStatement pStatement =
    connection.prepareStatement(sql,ResultSet.TYPE_SCROLL_INSENSITIVE,ResultSet.CONCUR_READ_ONLY);

    ResultSet scrollableResultSet = pStatement.executeQuery();

    System.out.println("Initial cursor position : "+ scrollableResultSet.getRow());
    scrollableResultSet.next();
    System.out.println("Current cursor position after moving cursor one step forward: "
                  + scrollableResultSet.getRow());
    System.out.println("Customer Id "+scrollableResultSet.getInt("custId"));
    System.out.println("Customer Name "+scrollableResultSet.getString("customerName"));
    System.out.println("Customer Name "+scrollableResultSet.getString("mailId"));
    System.out.println("\n");

} catch (SQLException exception) {
    System.out.println(exception.getMessage());
} catch (Exception exception) {
    System.out.println(exception.getMessage());
```



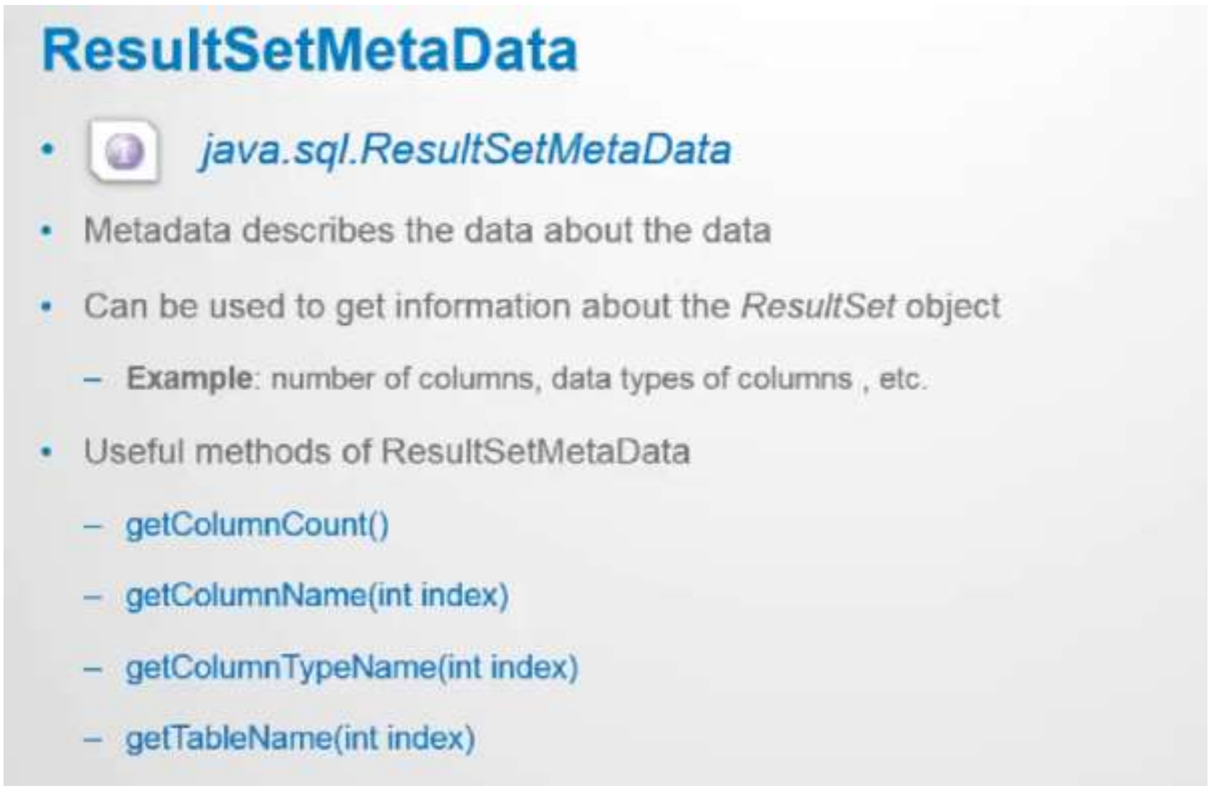
```
                    + scrollableResultSet.getRow());
        System.out.println("Customer Id "+scrollableResultSet.getInt("custId"));
        System.out.println("Customer Name "+scrollableResultSet.getString("customerName"));
        System.out.println("Customer Name "+scrollableResultSet.getString("mailId"));
        System.out.println("\n");

        scrollableResultSet.last();
        System.out.println("Current cursor position after moving cursor to the last postion:
                    + scrollableResultSet.getRow());
        System.out.println("Customer Id "+scrollableResultSet.getInt("custId"));
        System.out.println("Customer Name "+scrollableResultSet.getString("customerName"));
        System.out.println("Customer Name "+scrollableResultSet.getString("mailId"));
        System.out.println("\n");

    } catch (SQLException exception) {
        System.out.println(exception.getMessage());
    } catch (Exception exception) {
        System.out.println(exception.getMessage());
    }
  }
}
```



```
        scrollableResultSet.last();
        System.out.println("Current cursor position after moving cursor to the last postion:
                    + scrollableResultSet.getRow());
        System.out.println("Customer Id "+scrollableResultSet.getInt("custId"));
        System.out.println("Customer Name "+scrollableResultSet.getString("customerName"));
        System.out.println("Customer Name "+scrollableResultSet.getString("mailId"));
        System.out.println("\n");

        scrollableResultSet.previous();
        System.out.println("Current cursor position after moving cursor one step backward: "
                    + scrollableResultSet.getRow());
        System.out.println("Customer Id "+scrollableResultSet.getInt("custId"));
        System.out.println("Customer Name "+scrollableResultSet.getString("customerName"));
        System.out.println("Customer Name "+scrollableResultSet.getString("mailId"));
        System.out.println("\n");

    } catch (SQLException exception) {
        System.out.println(exception.getMessage());
    } catch (Exception exception) {
        System.out.println(exception.getMessage());
```

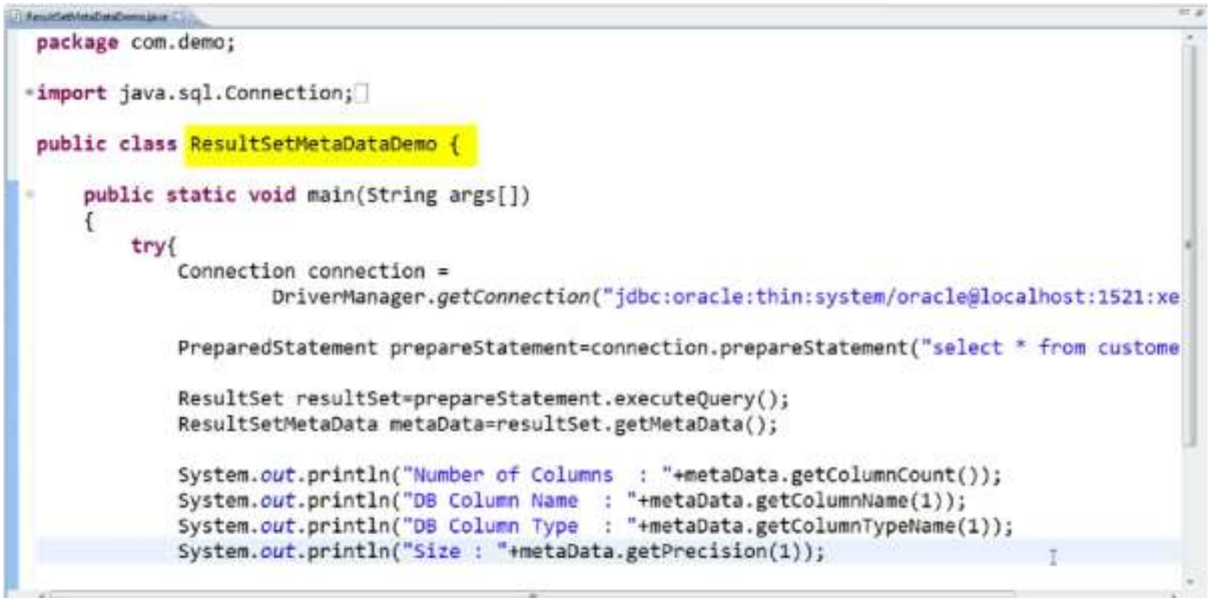


```
                    + scrollableResultSet.getRow());
        System.out.println("Customer Id "+scrollableResultSet.getInt("custId"));
        System.out.println("Customer Name "+scrollableResultSet.getString("customerName"));
        System.out.println("Customer Name "+scrollableResultSet.getString("mailId"));
        System.out.println("\n");


        scrollableResultSet.absolute(2);
        System.out.println("Current cursor position after executing the absolute method : "
                    + scrollableResultSet.getRow());
        System.out.println("Customer Id "+scrollableResultSet.getInt("custId"));
        System.out.println("Customer Name "+scrollableResultSet.getString("customerName"));
        System.out.println("Customer Name "+scrollableResultSet.getString("mailId"));

    } catch (SQLException exception) {
        System.out.println(exception.getMessage());
    } catch (Exception exception) {
        System.out.println(exception.getMessage());
    }
  }
}
```

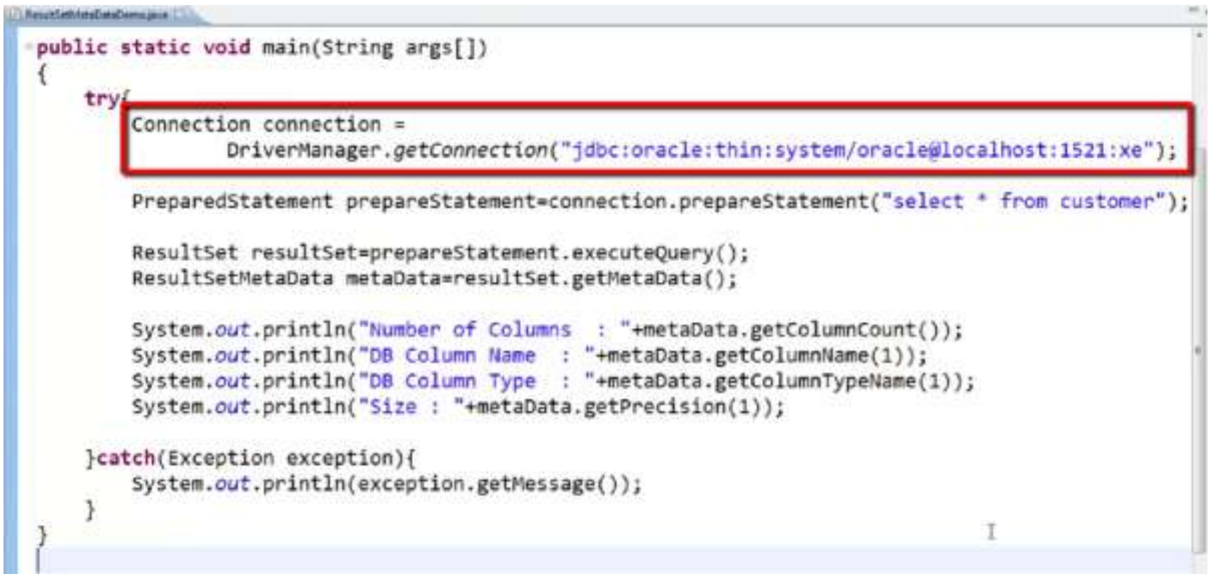


ResultSetMetaData

- *java.sql.ResultSetMetaData*
- Metadata describes the data about the data
- Can be used to get information about the *ResultSet* object
  - **Example**: number of columns, data types of columns , etc.
- Useful methods of ResultSetMetaData
  - getColumnCount()
  - getColumnName(int index)
  - getColumnTypeName(int index)
  - getTableName(int index)

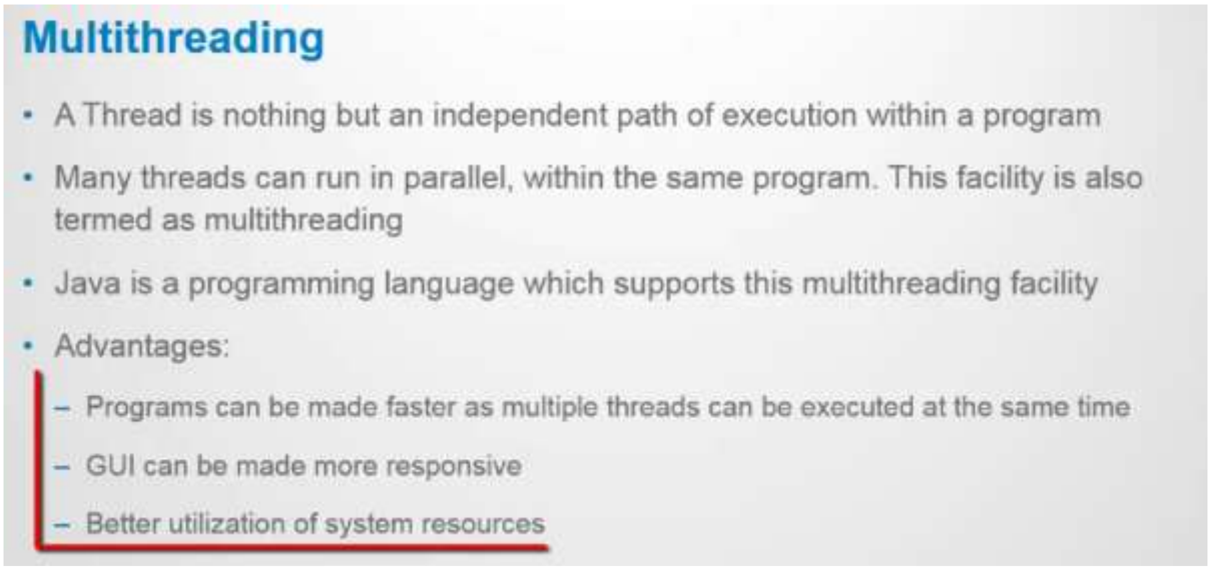


```java
package com.demo;

import java.sql.Connection;

public class ResultSetMetaDataDemo {

    public static void main(String args[])
    {
        try{
            Connection connection =
                    DriverManager.getConnection("jdbc:oracle:thin:system/oracle@localhost:1521:xe

            PreparedStatement prepareStatement=connection.prepareStatement("select * from custome

            ResultSet resultSet=prepareStatement.executeQuery();
            ResultSetMetaData metaData=resultSet.getMetaData();

            System.out.println("Number of Columns  : "+metaData.getColumnCount());
            System.out.println("DB Column Name   : "+metaData.getColumnName(1));
            System.out.println("DB Column Type   : "+metaData.getColumnTypeName(1));
            System.out.println("Size : "+metaData.getPrecision(1));
```



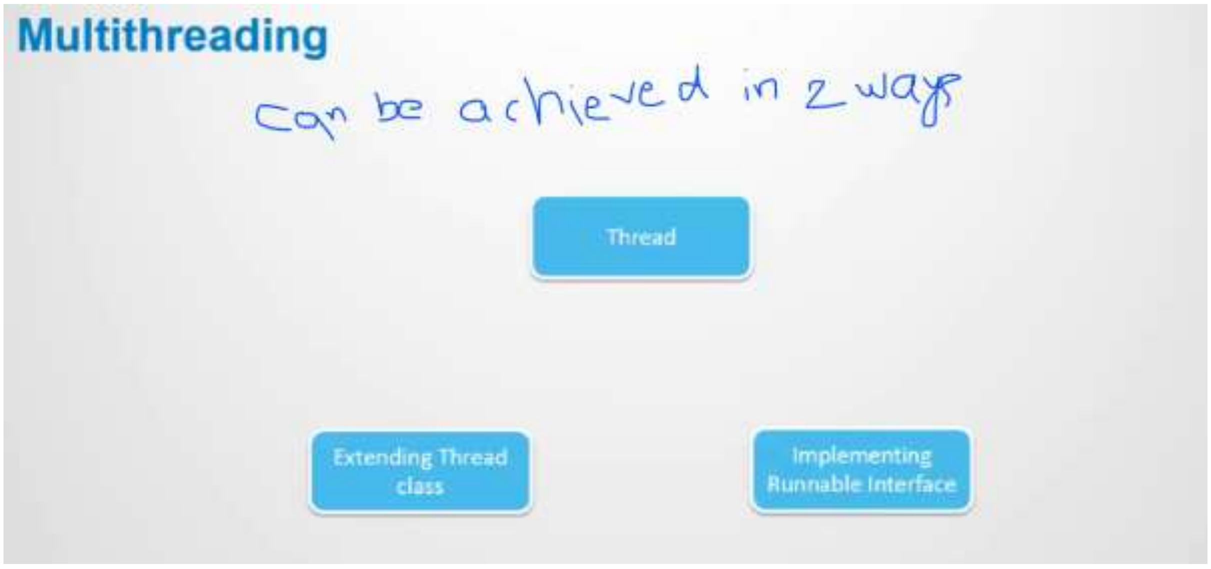
```
public static void main(String args[])
{
    try{
        Connection connection =
                DriverManager.getConnection("jdbc:oracle:thin:system/oracle@localhost:1521:xe");

        PreparedStatement prepareStatement=connection.prepareStatement("select * from customer");

        ResultSet resultSet=prepareStatement.executeQuery();
        ResultSetMetaData metaData=resultSet.getMetaData();

        System.out.println("Number of Columns  : "+metaData.getColumnCount());
        System.out.println("DB Column Name  : "+metaData.getColumnName(1));
        System.out.println("DB Column Type  : "+metaData.getColumnTypeName(1));
        System.out.println("Size : "+metaData.getPrecision(1));

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

MultiThreading:



Multithreading

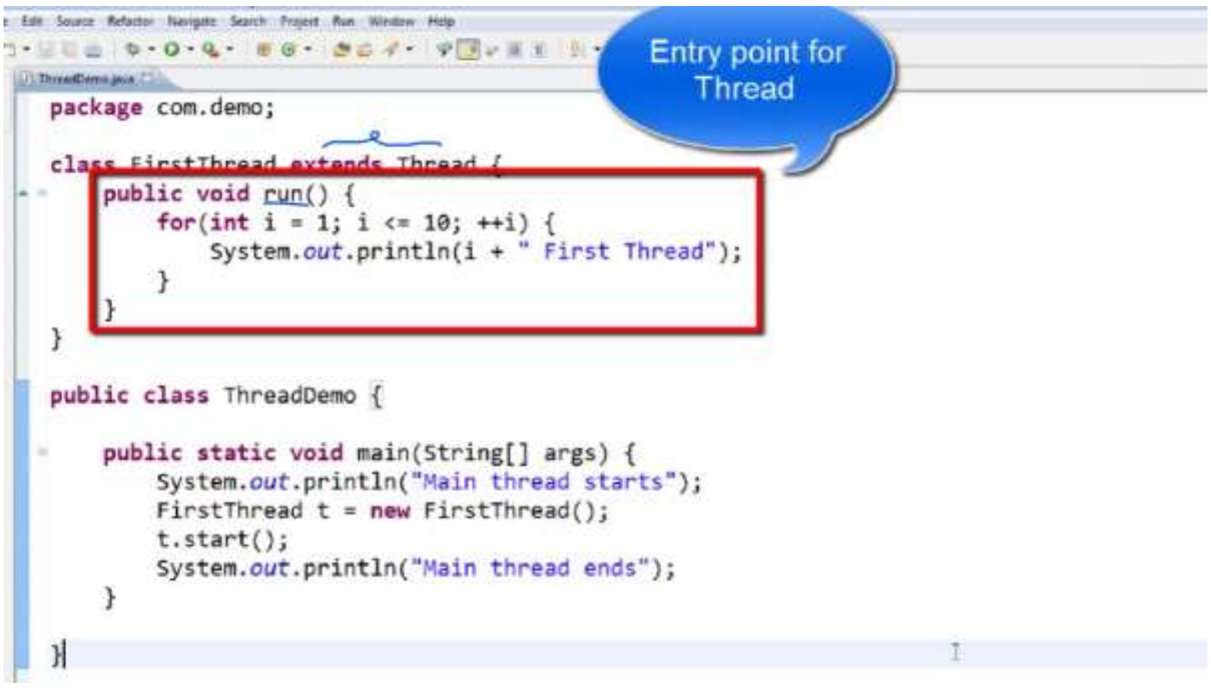
- A Thread is nothing but an independent path of execution within a program
- Many threads can run in parallel, within the same program. This facility is also termed as multithreading
- Java is a programming language which supports this multithreading facility
- Advantages:
  - Programs can be made faster as multiple threads can be executed at the same time
  - GUI can be made more responsive
  - Better utilization of system resources



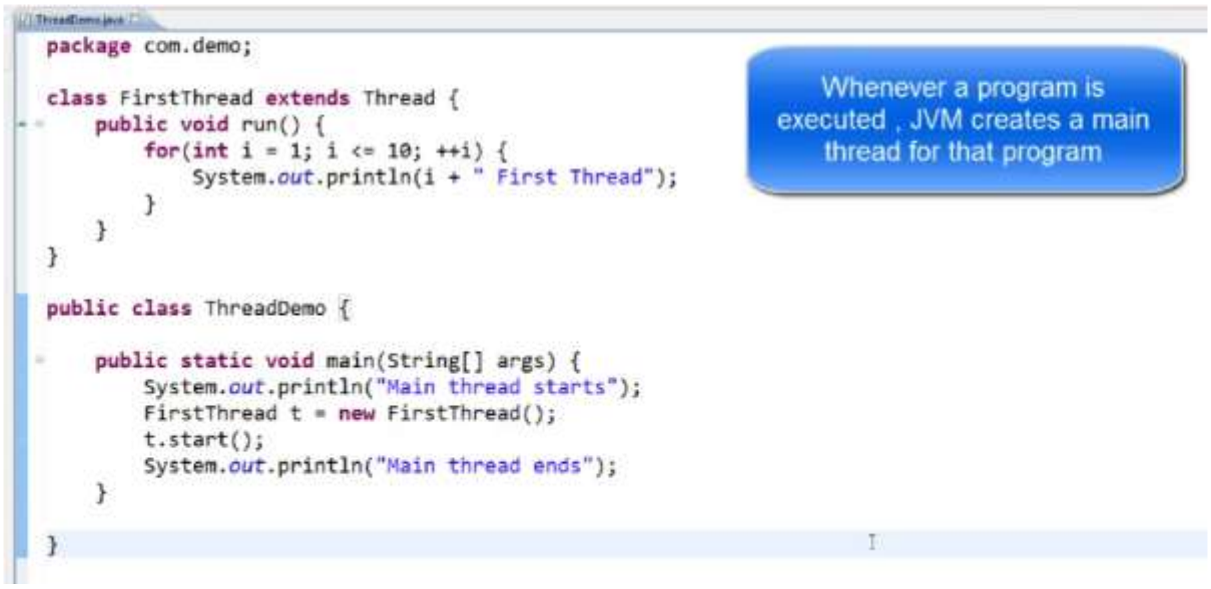
Multithreading

can be achieved in 2 ways

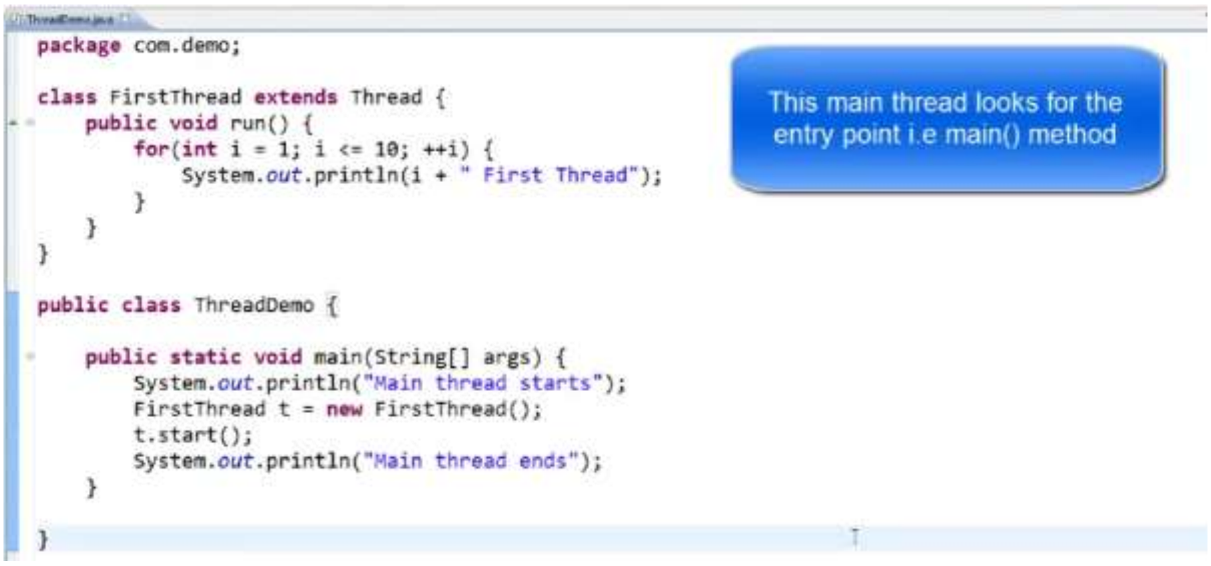
The class extending Thread must override run method , which is entry point from Thread



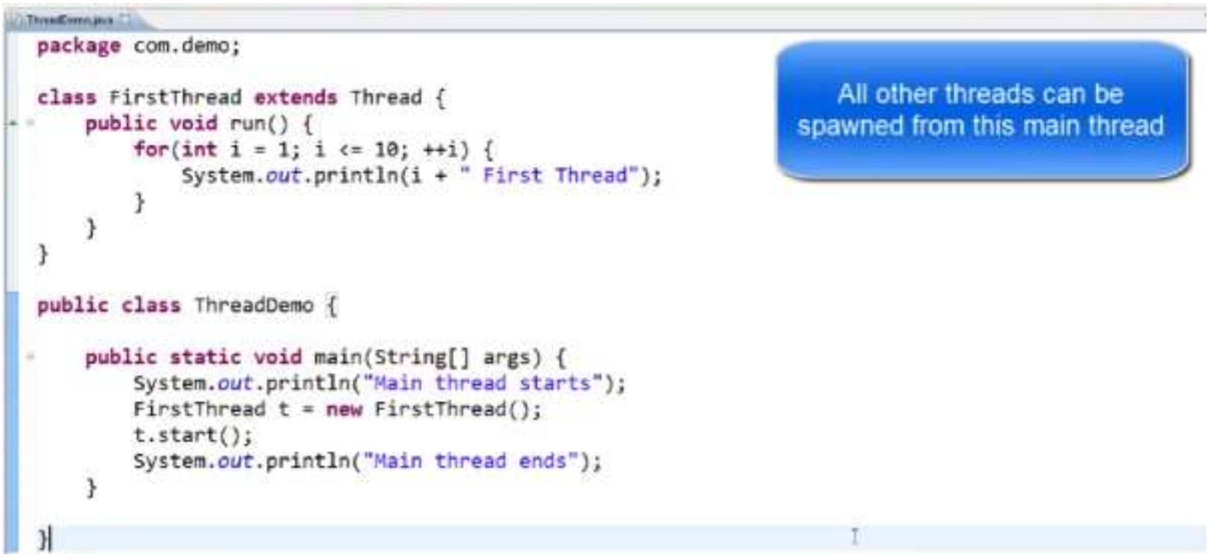
```
package com.demo;

class FirstThread extends Thread {
    public void run() {
        for(int i = 1; i <= 10; ++i) {
            System.out.println(i + " First Thread");
        }
    }
}

public class ThreadDemo {

    public static void main(String[] args) {
        System.out.println("Main thread starts");
        FirstThread t = new FirstThread();
        t.start();
        System.out.println("Main thread ends");
    }

}
```


Whenever a program is executed , JVM creates a main thread for that program

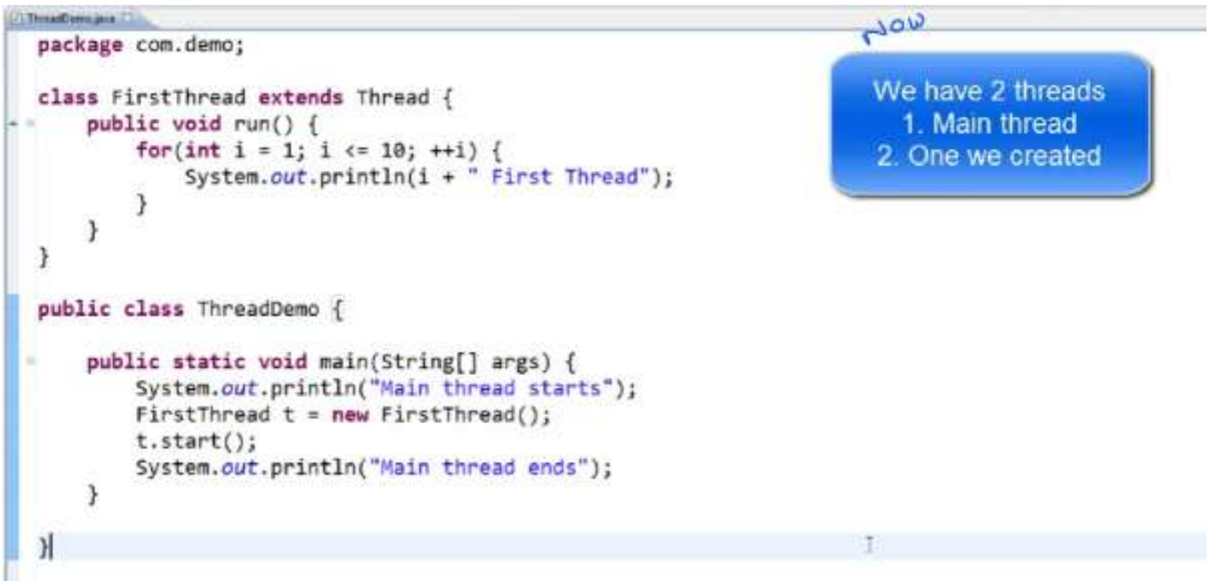
```
package com.demo;

class FirstThread extends Thread {
    public void run() {
        for(int i = 1; i <= 10; ++i) {
            System.out.println(i + " First Thread");
        }
    }
}

public class ThreadDemo {

    public static void main(String[] args) {
        System.out.println("Main thread starts");
        FirstThread t = new FirstThread();
        t.start();
        System.out.println("Main thread ends");
    }

}
```


This main thread looks for the entry point i.e main() method

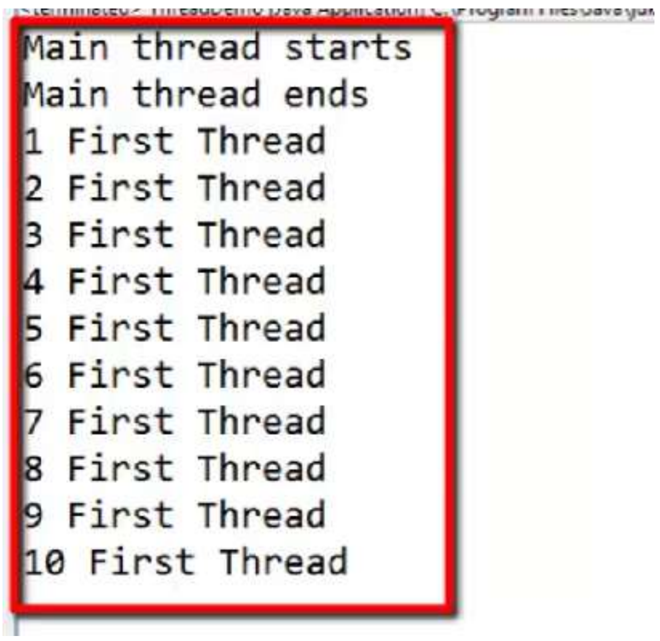
```
package com.demo;

class FirstThread extends Thread {
    public void run() {
        for(int i = 1; i <= 10; ++i) {
            System.out.println(i + " First Thread");
        }
    }
}

public class ThreadDemo {

    public static void main(String[] args) {
        System.out.println("Main thread starts");
        FirstThread t = new FirstThread();
        t.start();
        System.out.println("Main thread ends");
    }

}
```



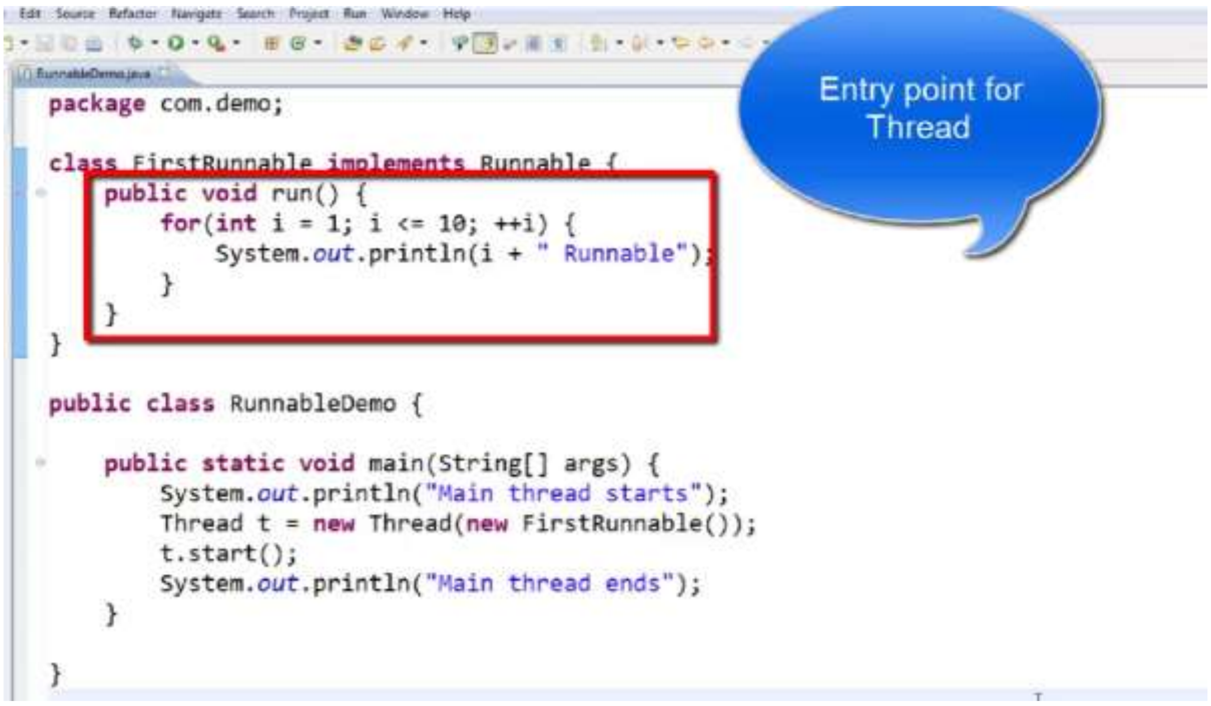
```
package com.demo;

class FirstThread extends Thread {
    public void run() {
        for(int i = 1; i <= 10; ++i) {
            System.out.println(i + " First Thread");
        }
    }
}

public class ThreadDemo {

    public static void main(String[] args) {
        System.out.println("Main thread starts");
        FirstThread t = new FirstThread();
        t.start();
        System.out.println("Main thread ends");
    }

}
```



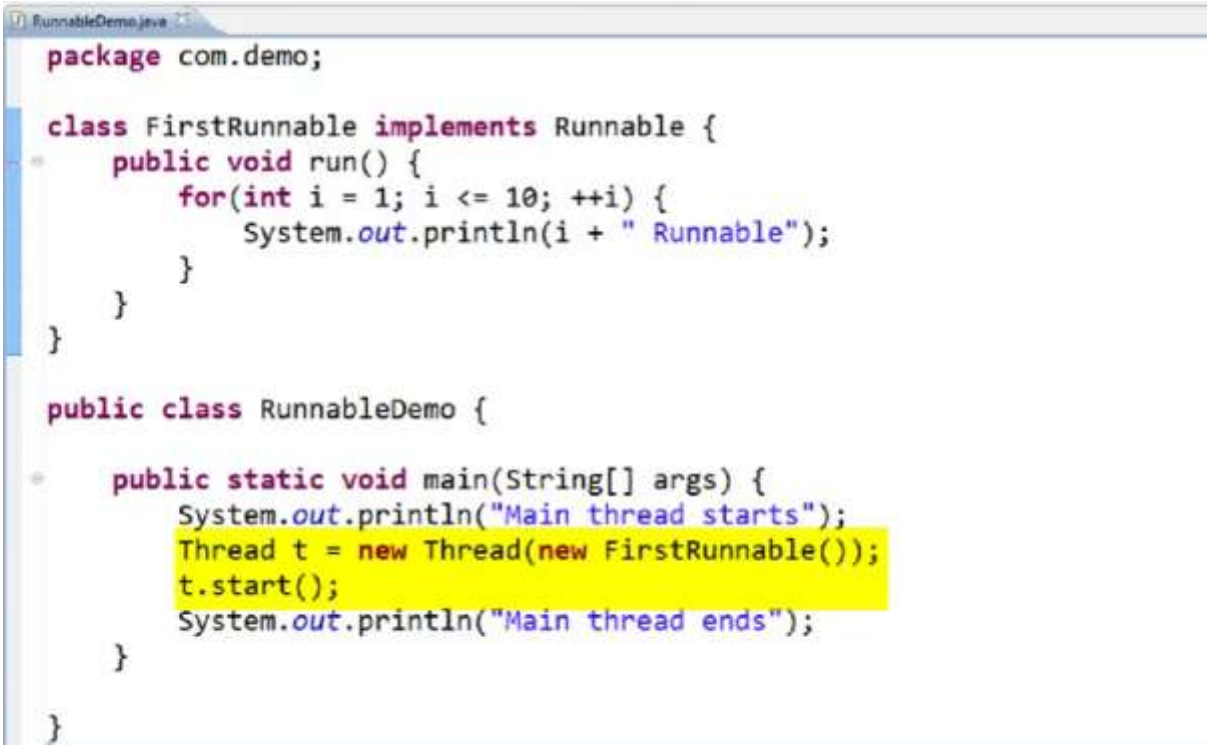
```
package com.demo;

class FirstThread extends Thread {
    public void run() {
        for(int i = 1; i <= 10; ++i) {
            System.out.println(i + " First Thread");
        }
    }
}

public class ThreadDemo {

    public static void main(String[] args) {
        System.out.println("Main thread starts");
        FirstThread t = new FirstThread();
        t.start();
        System.out.println("Main thread ends");
    }

}
```



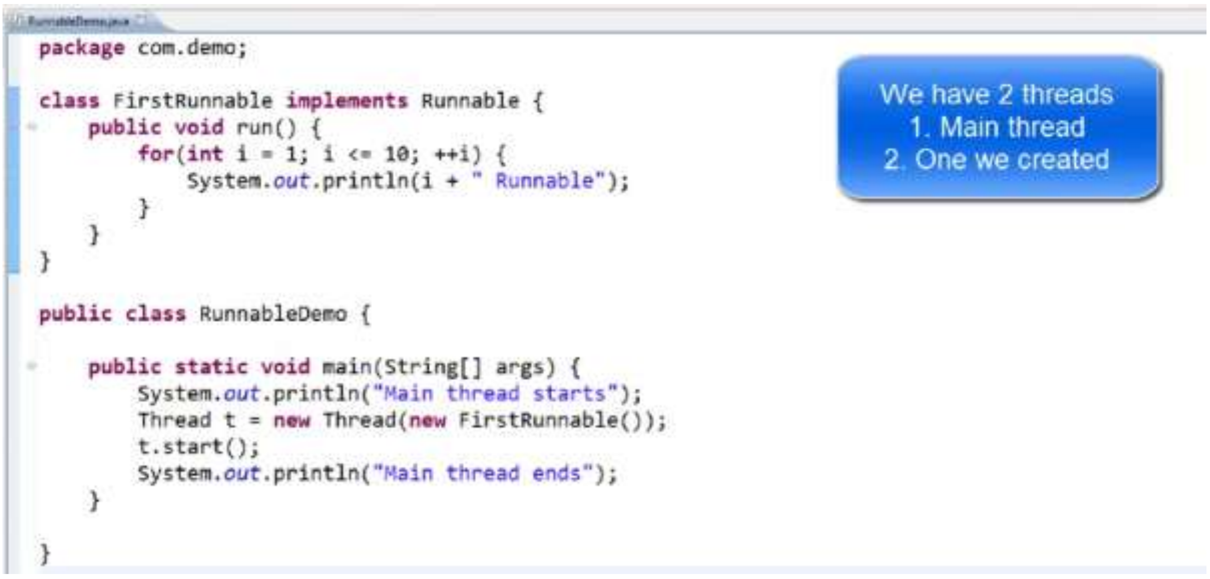
```
Main thread starts
Main thread ends
1 First Thread
2 First Thread
3 First Thread
4 First Thread
5 First Thread
6 First Thread
7 First Thread
8 First Thread
9 First Thread
10 First Thread
```


Edit Source Refactor Navigate Search Project Run Window Help
RunnableDemo.java
Entry point for Thread

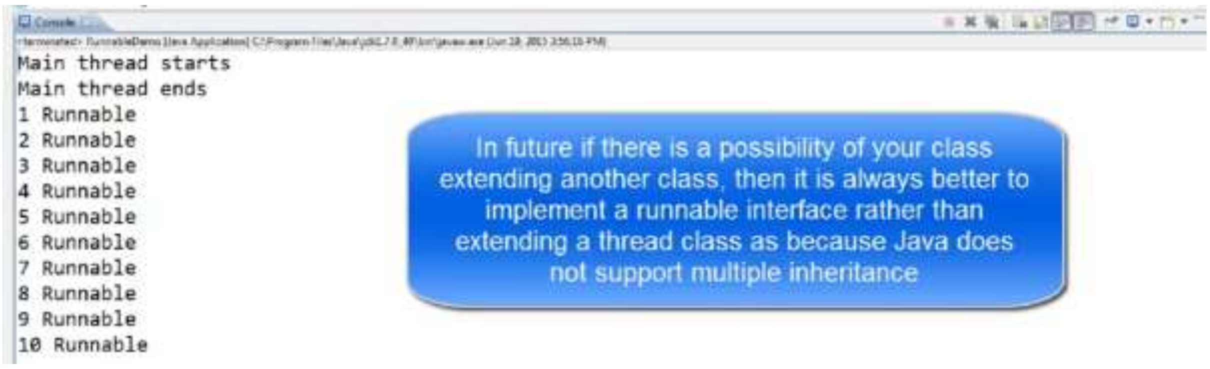
```
package com.demo;

class FirstRunnable implements Runnable {
    public void run() {
        for(int i = 1; i <= 10; ++i) {
            System.out.println(i + " Runnable");
        }
    }
}

public class RunnableDemo {

    public static void main(String[] args) {
        System.out.println("Main thread starts");
        Thread t = new Thread(new FirstRunnable());
        t.start();
        System.out.println("Main thread ends");
    }

}
```



```
package com.demo;

class FirstRunnable implements Runnable {
    public void run() {
        for(int i = 1; i <= 10; ++i) {
            System.out.println(i + " Runnable");
        }
    }
}

public class RunnableDemo {

    public static void main(String[] args) {
        System.out.println("Main thread starts");
        Thread t = new Thread(new FirstRunnable());
        t.start();
        System.out.println("Main thread ends");
    }

}
```

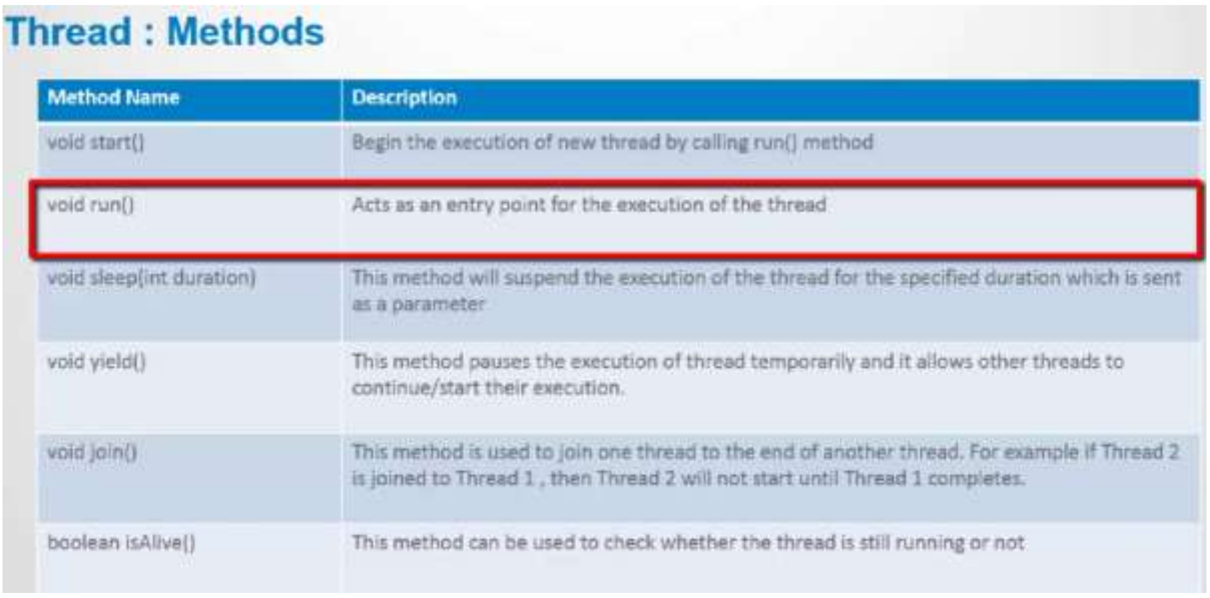


```
package com.demo;

class FirstRunnable implements Runnable {
    public void run() {
        for(int i = 1; i <= 10; ++i) {
            System.out.println(i + " Runnable");
        }
    }
}

public class RunnableDemo {

    public static void main(String[] args) {
        System.out.println("Main thread starts");
        Thread t = new Thread(new FirstRunnable());
        t.start();
        System.out.println("Main thread ends");
    }

}
```



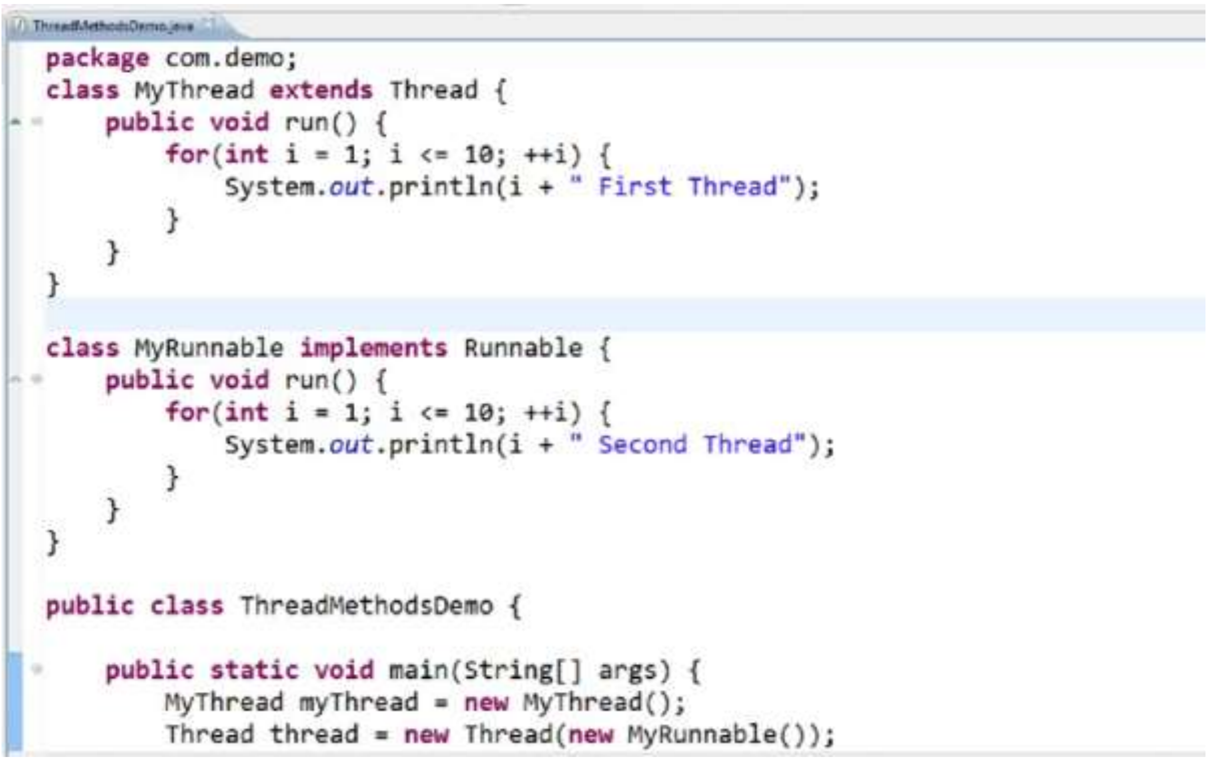
```
<terminated> RunnableDemo [Java Application] C:\Program Files\Java\jdk1.7.0_40\bin\javaw.exe (Jun 18, 2013 3:56:15 PM)
Main thread starts
Main thread ends
1 Runnable
2 Runnable
3 Runnable
4 Runnable
5 Runnable
6 Runnable
7 Runnable
8 Runnable
9 Runnable
10 Runnable
```

In future if there is a possibility of your class extending another class, then it is always better to implement a runnable interface rather than extending a thread class as because Java does not support multiple inheritance

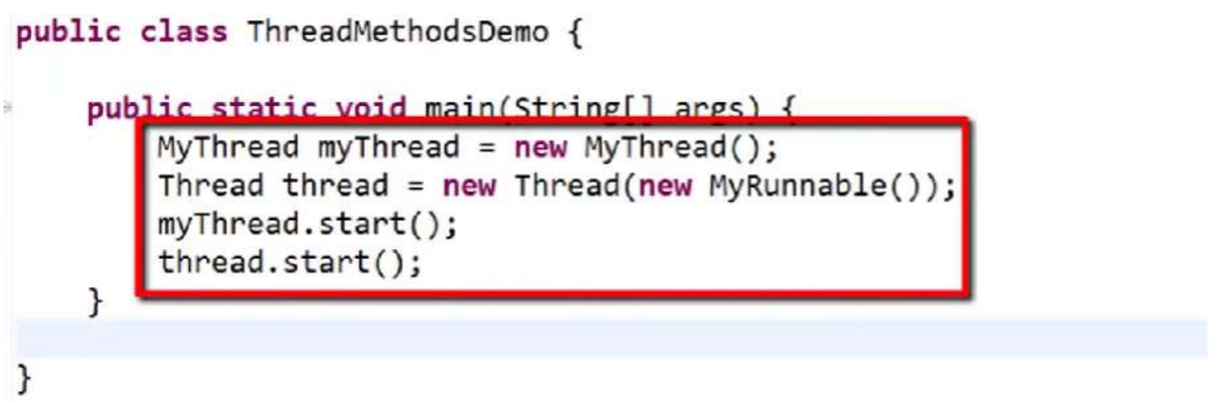


Thread : Methods

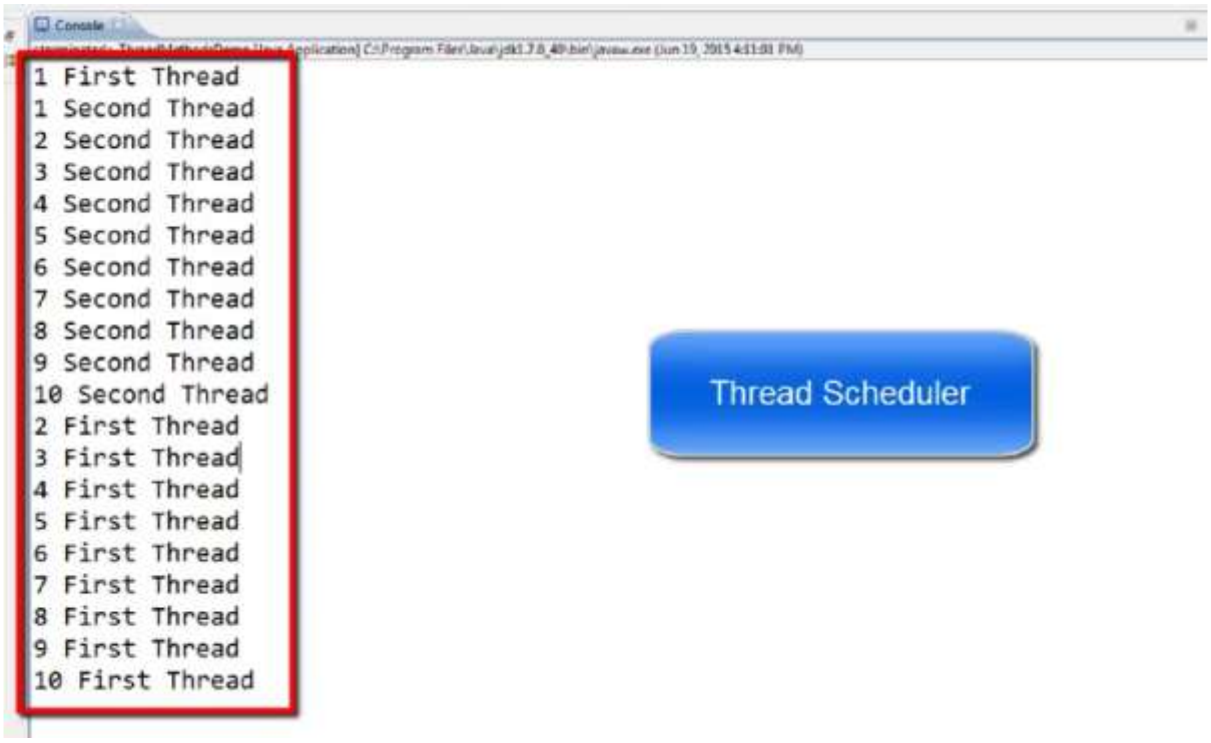
Method Name	Description
void start()	Begin the execution of new thread by calling run() method
void run()	Acts as an entry point for the execution of the thread
void sleep(int duration)	This method will suspend the execution of the thread for the specified duration which is sent as a parameter
void yield()	This method pauses the execution of thread temporarily and it allows other threads to continue/start their execution.
void join()	This method is used to join one thread to the end of another thread. For example if Thread 2 is joined to Thread 1 , then Thread 2 will not start until Thread 1 completes.
boolean isAlive()	This method can be used to check whether the thread is still running or not



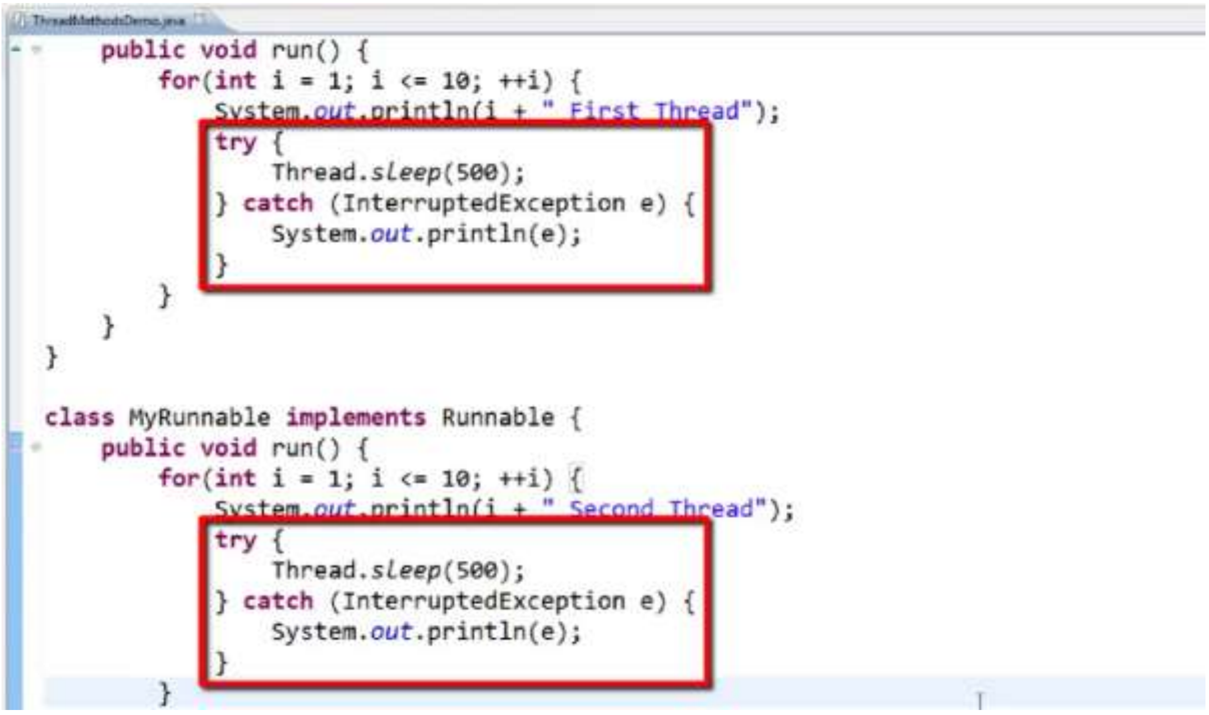
```
package com.demo;
class MyThread extends Thread {
    public void run() {
        for(int i = 1; i <= 10; ++i) {
            System.out.println(i + " First Thread");
        }
    }
}

class MyRunnable implements Runnable {
    public void run() {
        for(int i = 1; i <= 10; ++i) {
            System.out.println(i + " Second Thread");
        }
    }
}

public class ThreadMethodsDemo {

    public static void main(String[] args) {
        MyThread myThread = new MyThread();
        Thread thread = new Thread(new MyRunnable());
```



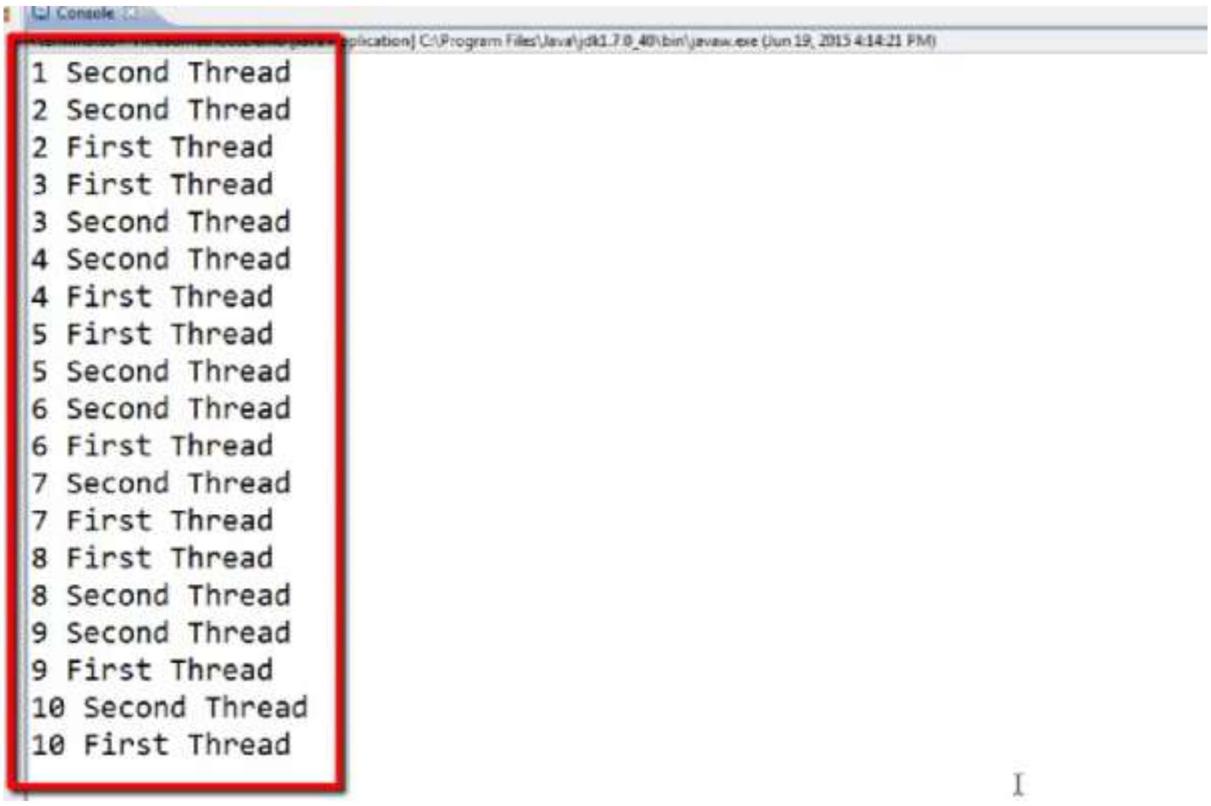
```
public class ThreadMethodsDemo {

    public static void main(String[] args) {
        MyThread myThread = new MyThread();
        Thread thread = new Thread(new MyRunnable());
        myThread.start();
        thread.start();
    }

}
```



```
<terminated> ThreadMethodsDemo [Java Application] C:\Program Files\Java\jdk1.7.0_40\bin\javaw.exe (Jun 19, 2015 4:31:03 PM)
1 First Thread
1 Second Thread
2 Second Thread
3 Second Thread
4 Second Thread
5 Second Thread
6 Second Thread
7 Second Thread
8 Second Thread
9 Second Thread
10 Second Thread
2 First Thread
3 First Thread
4 First Thread
5 First Thread
6 First Thread
7 First Thread
8 First Thread
9 First Thread
10 First Thread
```

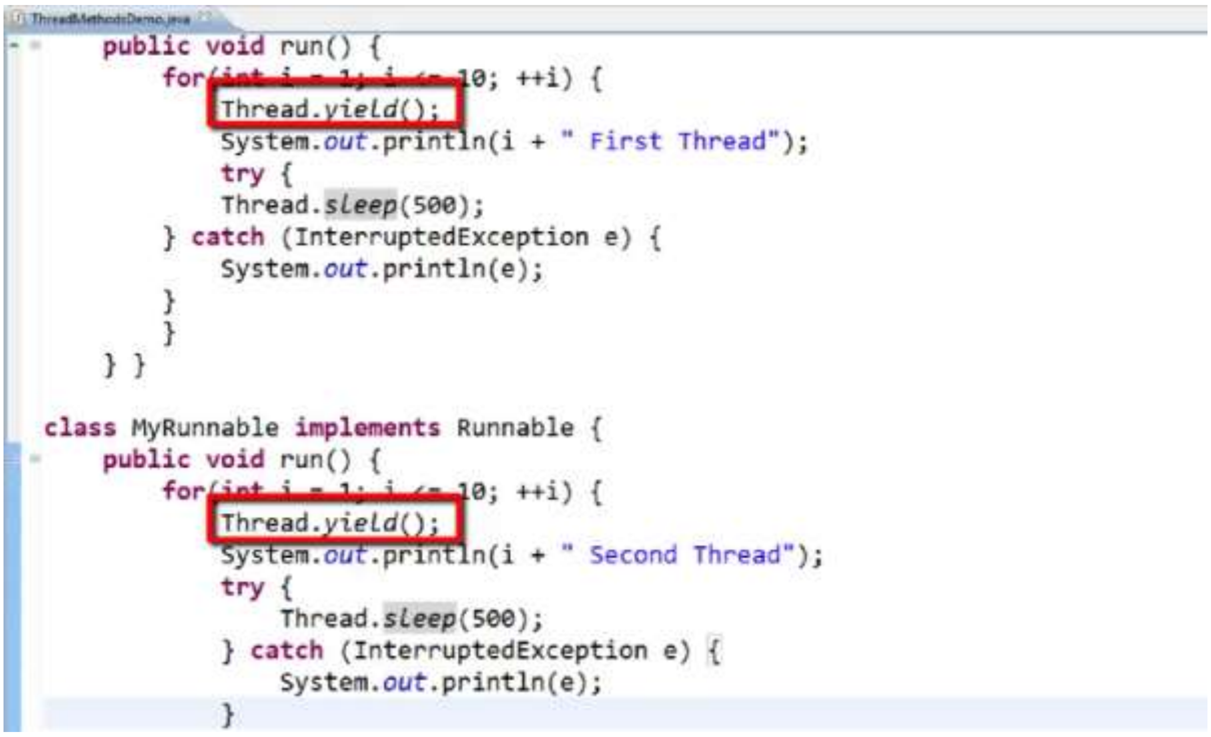


```java
    public void run() {
        for(int i = 1; i <= 10; ++i) {
            System.out.println(i + " First Thread");
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}

class MyRunnable implements Runnable {
    public void run() {
        for(int i = 1; i <= 10; ++i) {
            System.out.println(i + " Second Thread");
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
```



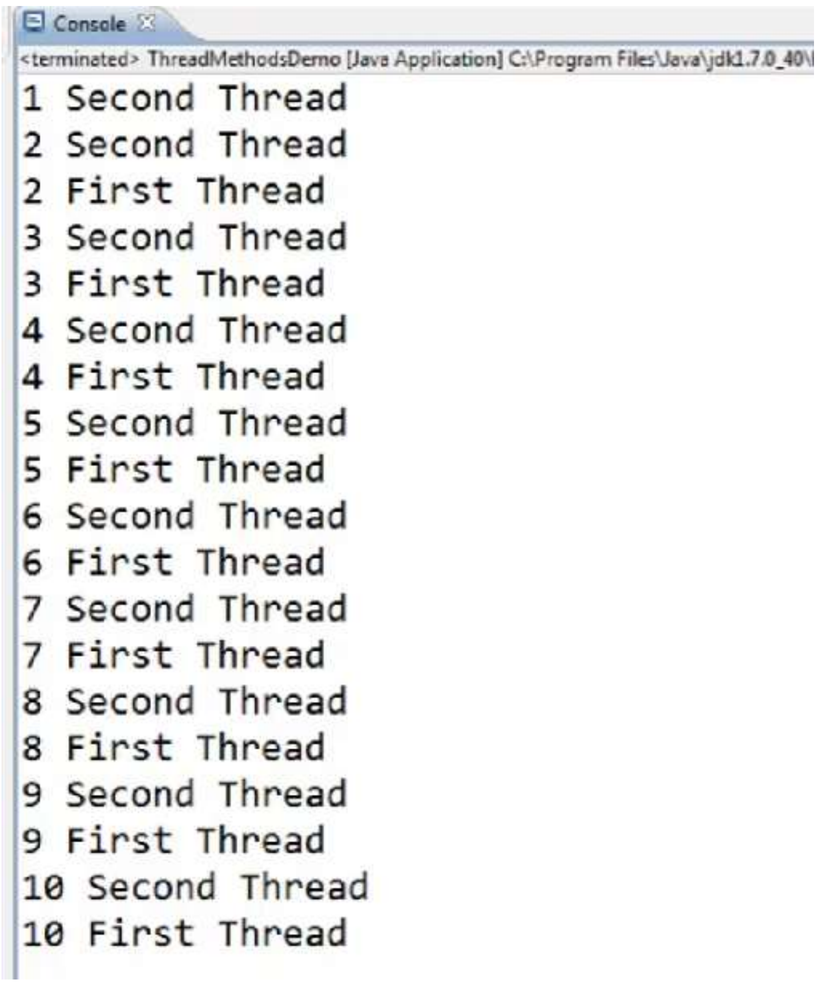
Console

```
plication] C:\Program Files\Java\jdk1.7.0_40\bin\javaw.exe (Jun 19, 2015 4:14:21 PM)
1 Second Thread
2 Second Thread
2 First Thread
3 First Thread
3 Second Thread
4 Second Thread
4 First Thread
5 First Thread
5 Second Thread
6 Second Thread
6 First Thread
7 Second Thread
7 First Thread
8 First Thread
8 Second Thread
9 Second Thread
9 First Thread
10 Second Thread
10 First Thread
```



ThreadMethodsDemo.java

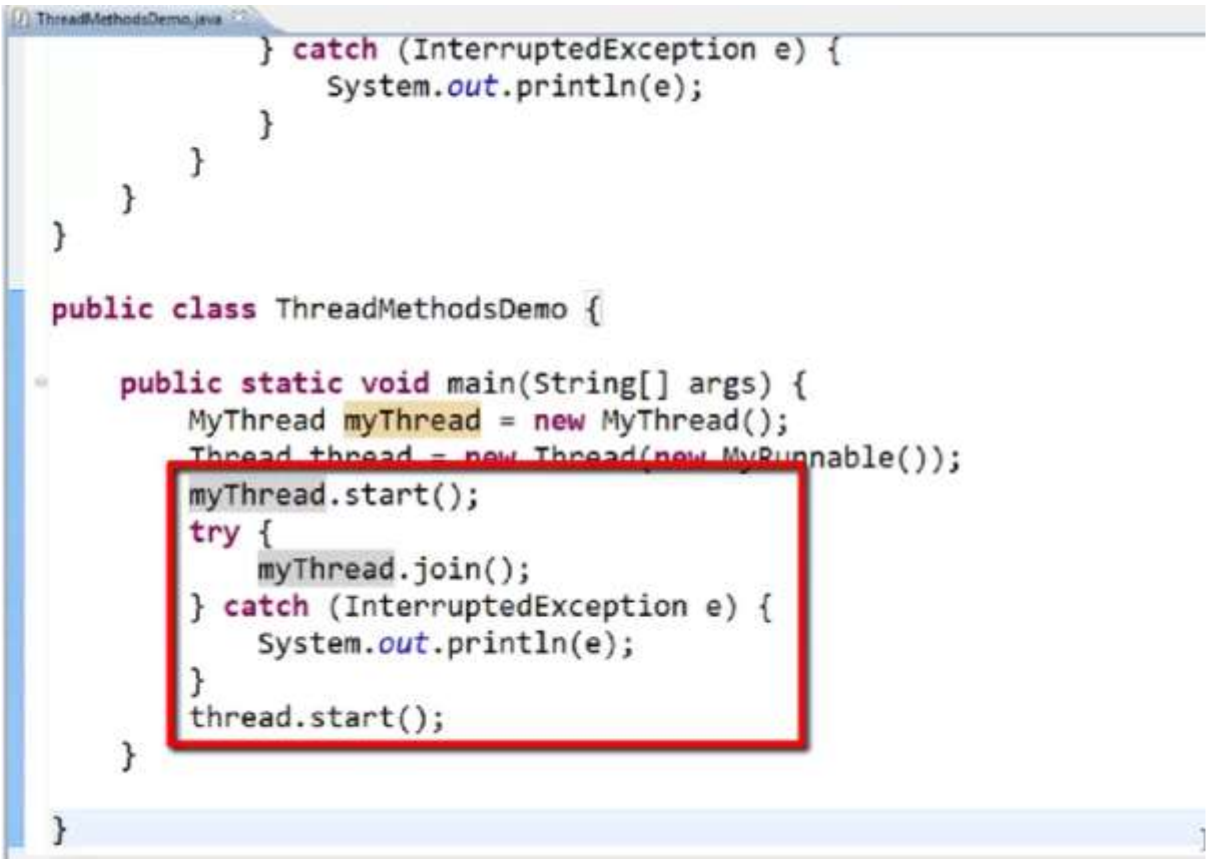
```java
    public void run() {
        for(int i = 1; i <= 10; ++i) {
            Thread.yield();
            System.out.println(i + " First Thread");
            try {
            Thread.sleep(500);
        } catch (InterruptedException e) {
            System.out.println(e);
        }
        }
    } }

class MyRunnable implements Runnable {
    public void run() {
        for(int i = 1; i <= 10; ++i) {
            Thread.yield();
            System.out.println(i + " Second Thread");
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
```



Console

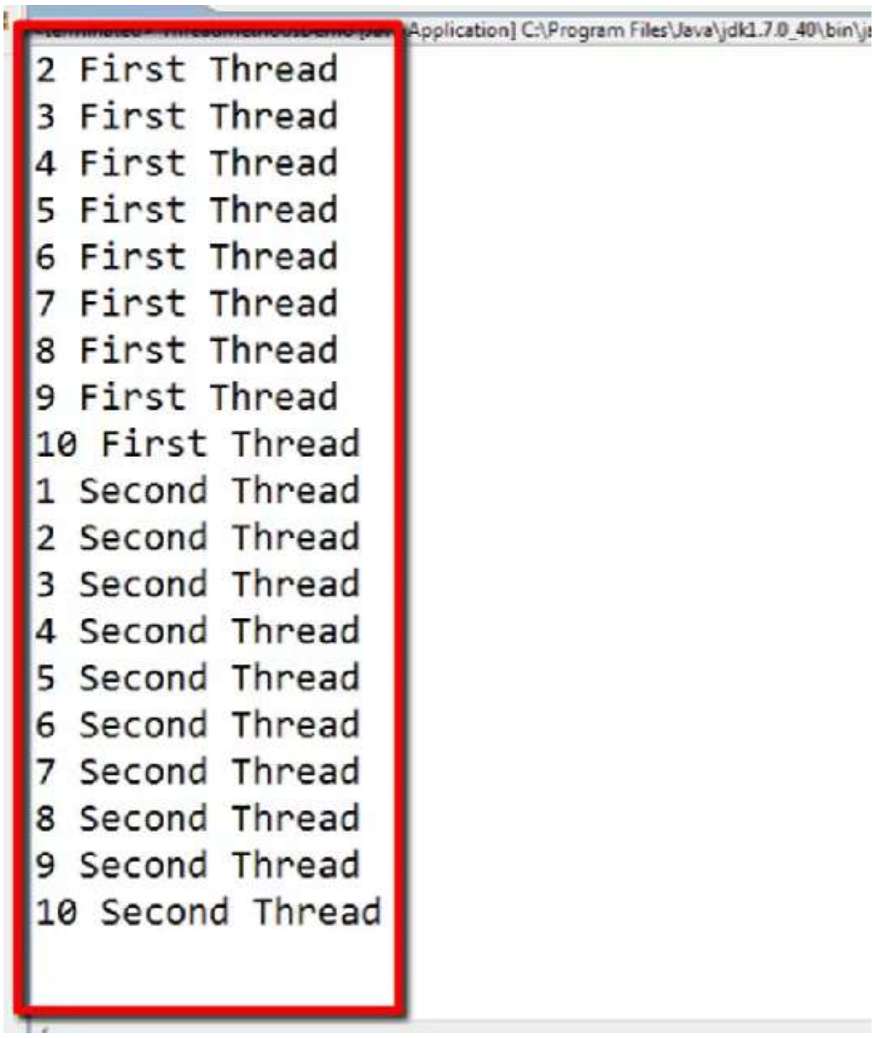
<terminated> ThreadMethodsDemo [Java Application] C:\Program Files\Java\jdk1.7.0_40\

```
1 Second Thread
2 Second Thread
2 First Thread
3 Second Thread
3 First Thread
4 Second Thread
4 First Thread
5 Second Thread
5 First Thread
6 Second Thread
6 First Thread
7 Second Thread
7 First Thread
8 Second Thread
8 First Thread
9 Second Thread
9 First Thread
10 Second Thread
10 First Thread
```



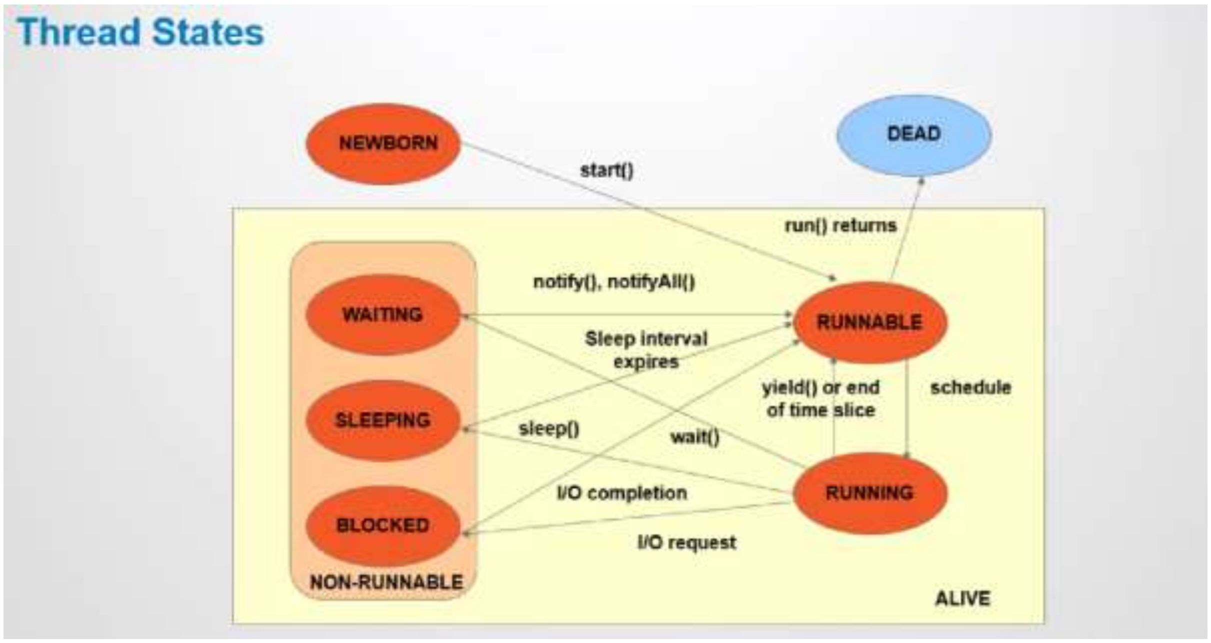
ThreadMethodsDemo.java

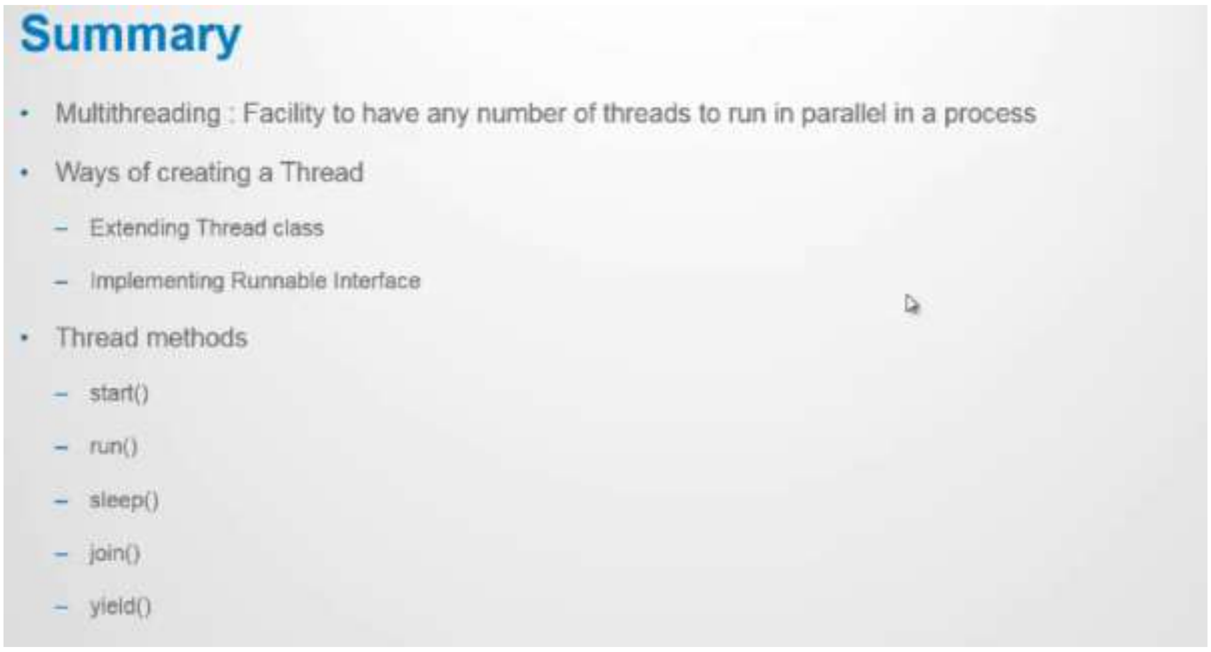
```
                } catch (InterruptedException e) {
                    System.out.println(e);
                }
            }
        }
    }

    public class ThreadMethodsDemo {

        public static void main(String[] args) {
            MyThread myThread = new MyThread();
            Thread thread = new Thread(new MyRunnable());
            myThread.start();
            try {
                myThread.join();
            } catch (InterruptedException e) {
                System.out.println(e);
            }
            thread.start();
        }

    }
```



```
Application] C:\Program Files\Java\jdk1.7.0_40\bin\j
2 First Thread
3 First Thread
4 First Thread
5 First Thread
6 First Thread
7 First Thread
8 First Thread
9 First Thread
10 First Thread
1 Second Thread
2 Second Thread
3 Second Thread
4 Second Thread
5 Second Thread
6 Second Thread
7 Second Thread
8 Second Thread
9 Second Thread
10 Second Thread
```

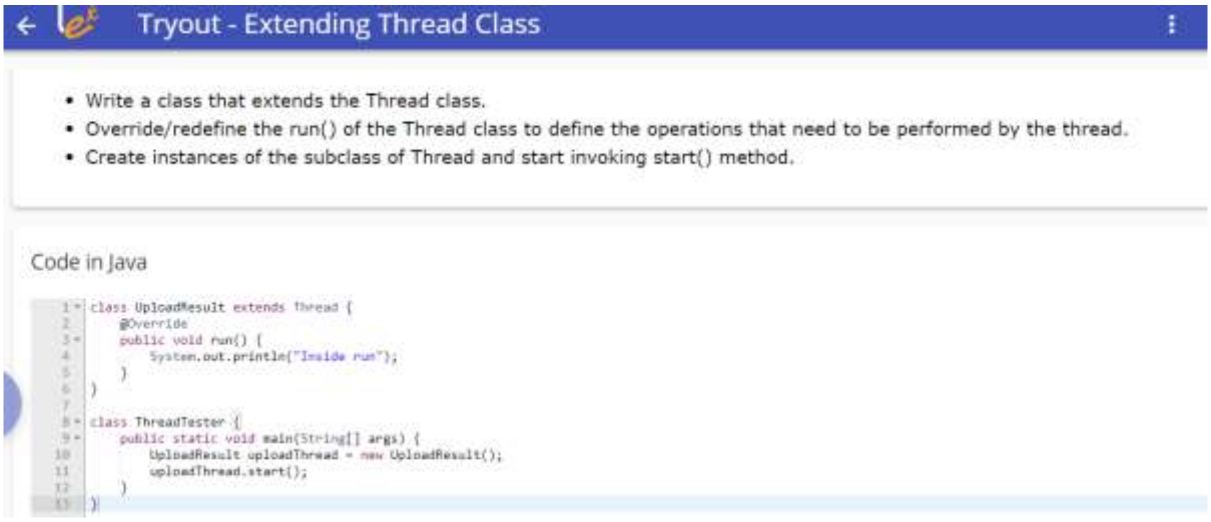
Thread States





Summary

- Multithreading : Facility to have any number of threads to run in parallel in a process
- Ways of creating a Thread
  - Extending Thread class
  - Implementing Runnable Interface
- Thread methods
  - start()
  - run()
  - sleep()
  - join()
  - yield()

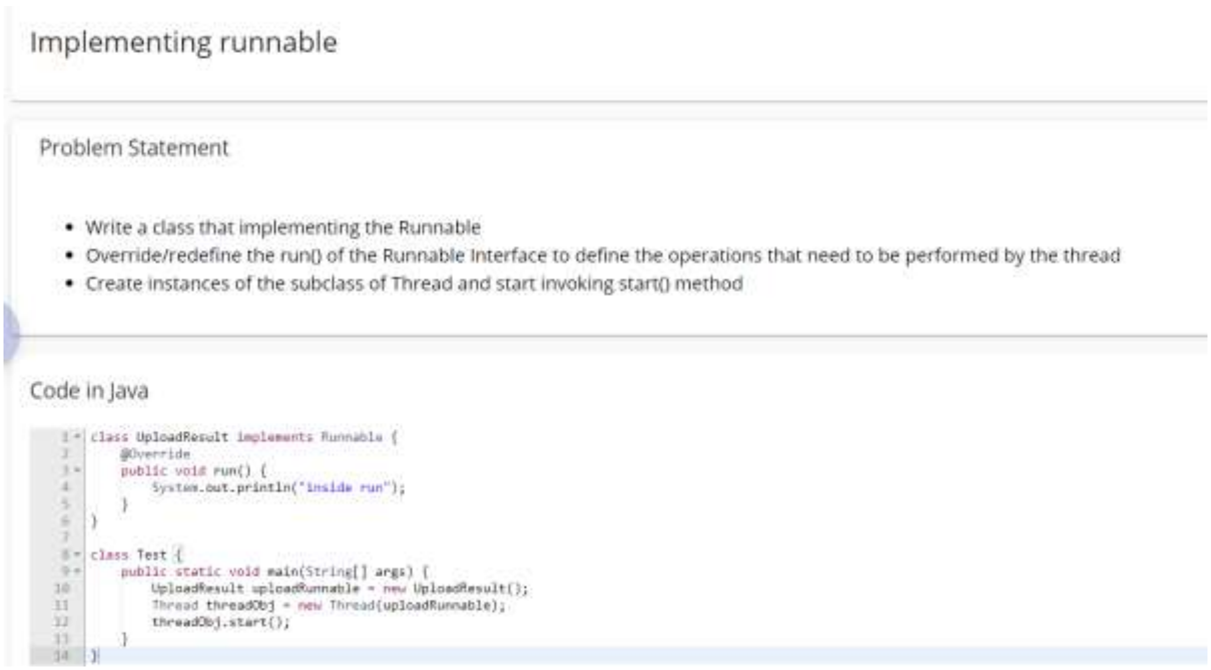


Tryout - Extending Thread Class

- Write a class that extends the Thread class.
- Override/redefine the run() of the Thread class to define the operations that need to be performed by the thread.
- Create instances of the subclass of Thread and start invoking start() method.

Code in Java

```java
class UploadResult extends Thread {
    @Override
    public void run() {
        System.out.println("Inside run");
    }
}

class ThreadTester {
    public static void main(String[] args) {
        UploadResult uploadThread = new UploadResult();
        uploadThread.start();
    }
}
```



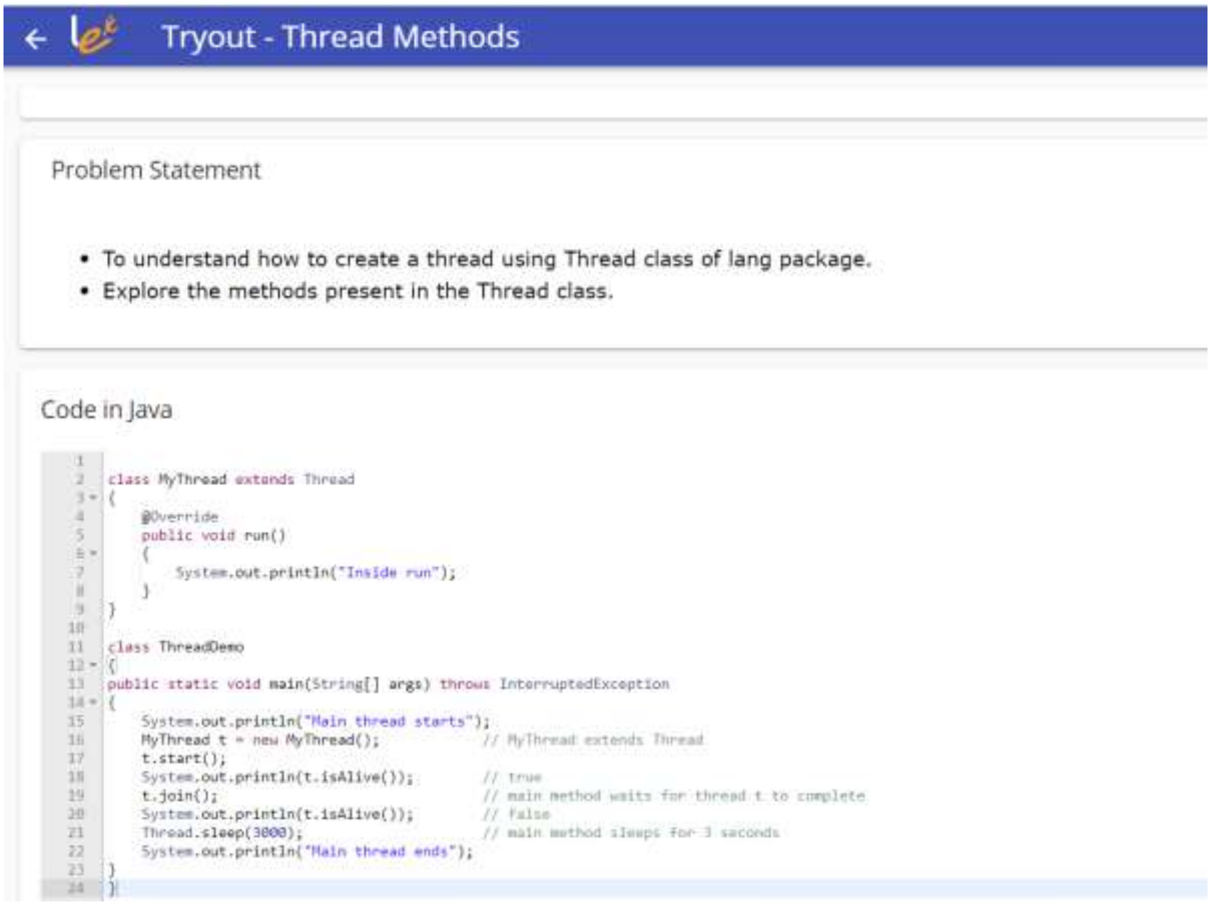
Implementing runnable

Problem Statement

- Write a class that implementing the Runnable
- Override/redefine the run() of the Runnable Interface to define the operations that need to be performed by the thread
- Create instances of the subclass of Thread and start invoking start() method

Code in Java

```java
class UploadResult implements Runnable {
    @Override
    public void run() {
        System.out.println("inside run");
    }
}

class Test {
    public static void main(String[] args) {
        UploadResult uploadRunnable = new UploadResult();
        Thread threadObj = new Thread(uploadRunnable);
        threadObj.start();
    }
}
```



← Tryout - Thread Methods

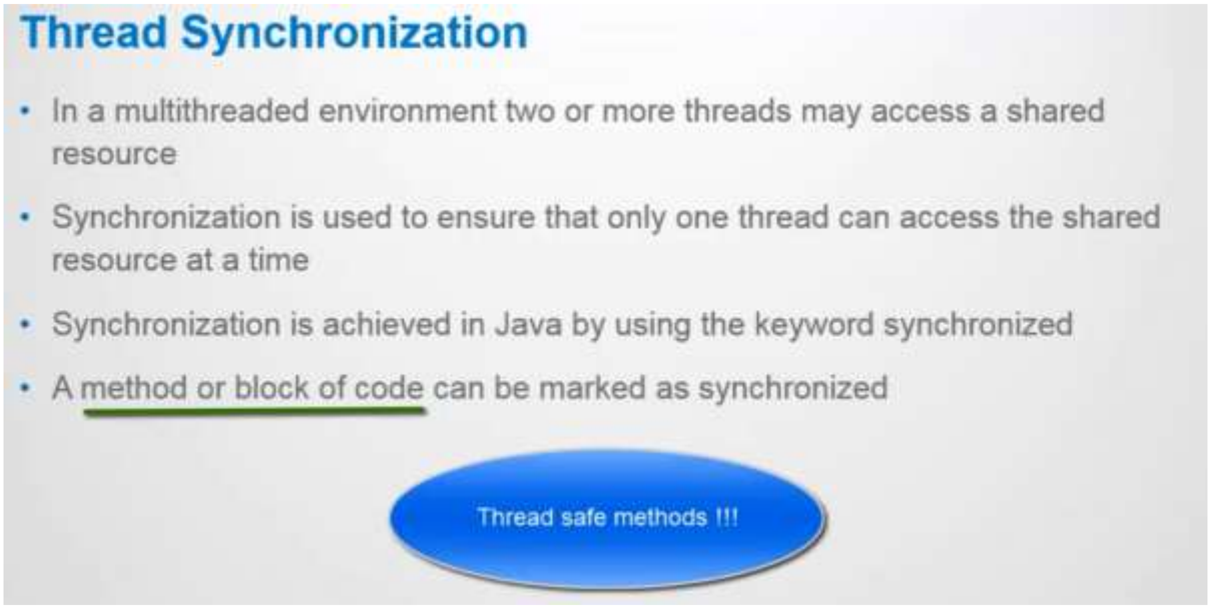
Problem Statement

- To understand how to create a thread using Thread class of lang package.
- Explore the methods present in the Thread class.

Code in Java

```java

class MyThread extends Thread
{
    @Override
    public void run()
    {
        System.out.println("Inside run");
    }
}

class ThreadDemo
{
public static void main(String[] args) throws InterruptedException
{
    System.out.println("Main thread starts");
    MyThread t = new MyThread();            // MyThread extends Thread
    t.start();
    System.out.println(t.isAlive());        // true
    t.join();                               // main method waits for thread t to complete
    System.out.println(t.isAlive());        // False
    Thread.sleep(3000);                     // main method sleeps for 3 seconds
    System.out.println("Main thread ends");
}
}
```

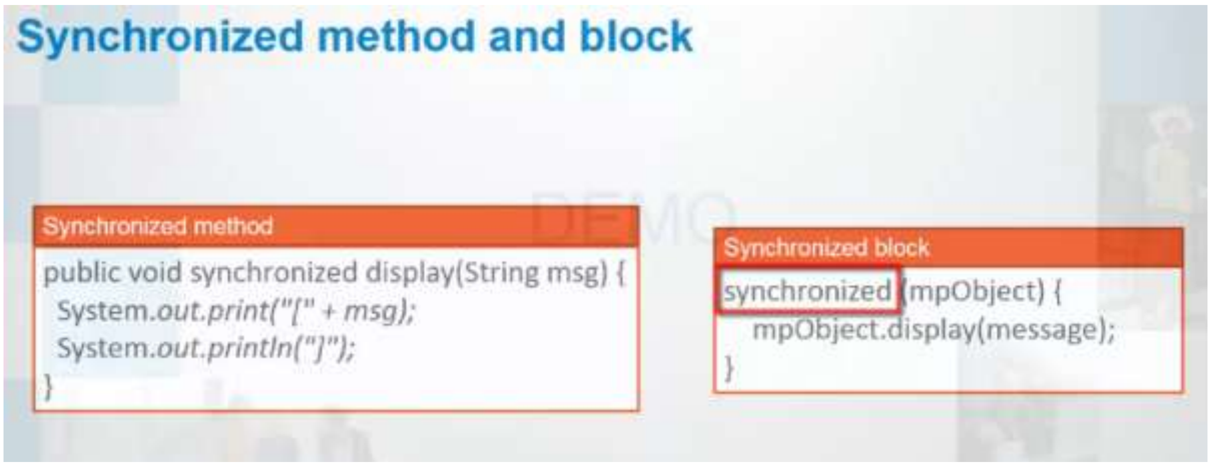
```
//Thread Scheduling and Priority


class Thread1 extends Thread {
      @Override
      public void run() {
          System.out.println("inside Thread1");
            for(int i=0;i<3;i++)
            {
                System.out.println("inside Thread1: "+i);
            }
            System.out.println("Thread1 finished");
      }
}

class Thread2 extends Thread {
      @Override
      public void run() {
            System.out.println("inside Thread2");
            for(int i=0;i<3;i++)
            {
                System.out.println("inside Thread2: "+i);
            }
            System.out.println("Thread2 finished");
      }
}

class ThreadTester {
      public static void main(String args[]) throws Exception {
            Thread1 thread1 = new Thread1();
            Thread2 thread2 = new Thread2();
            thread1.setPriority(Thread.MIN_PRIORITY);
            thread2.setPriority(Thread.MAX_PRIORITY);
            thread1.start();
            thread2.start();
      }
}
```



Thread Synchronization

- In a multithreaded environment two or more threads may access a shared resource
- Synchronization is used to ensure that only one thread can access the shared resource at a time
- Synchronization is achieved in Java by using the keyword synchronized
- A method or block of code can be marked as synchronized

Thread safe methods !!!



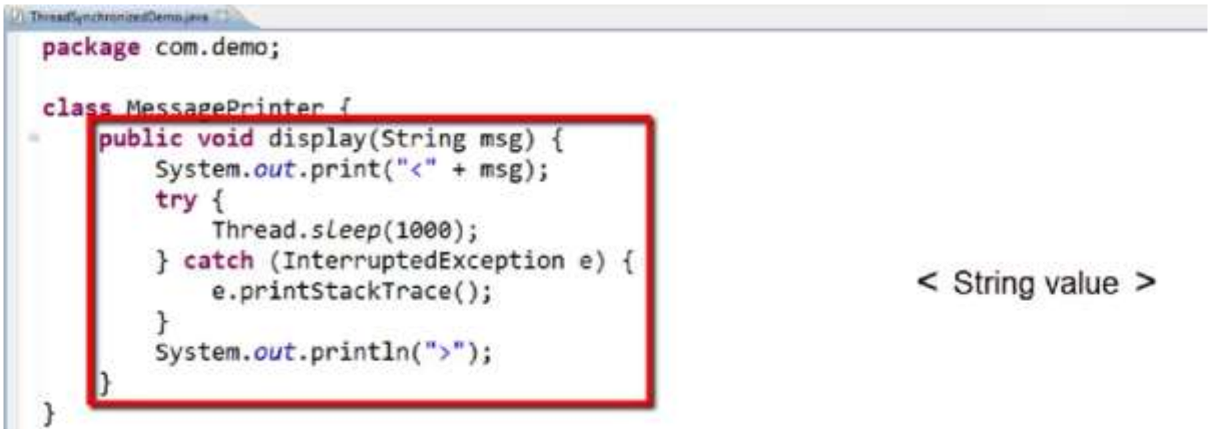
Synchronized method and block

Synchronized method

```
public void synchronized display(String msg) {
 System.out.print("[" + msg);
 System.out.println("]");
}
```

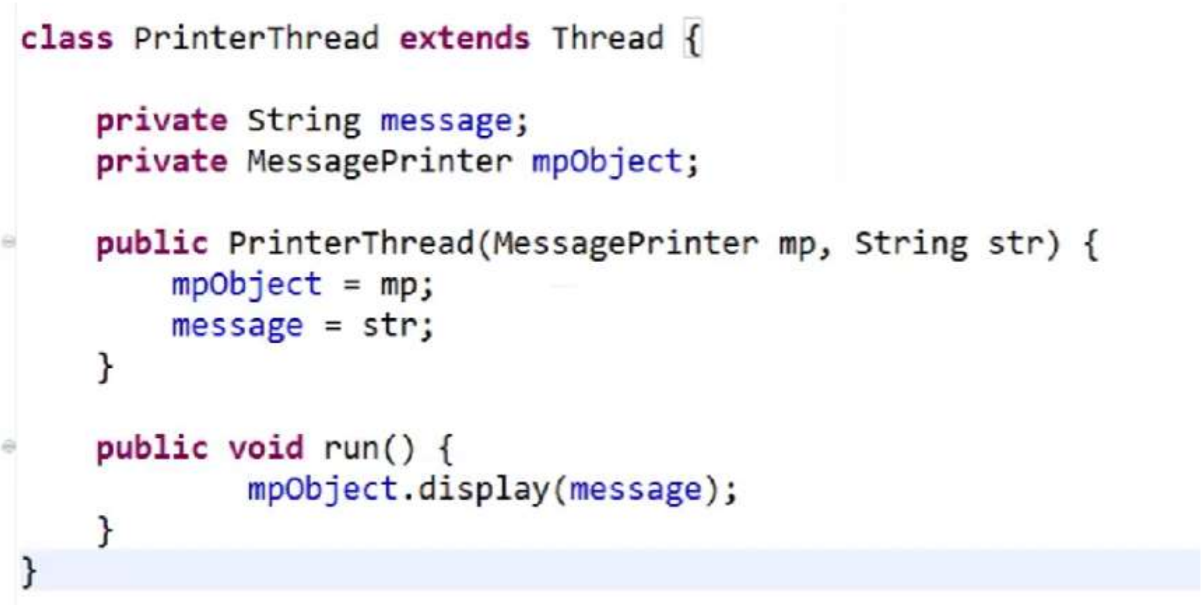
Synchronized block

```
synchronized (mpObject) {
  mpObject.display(message);
}
```

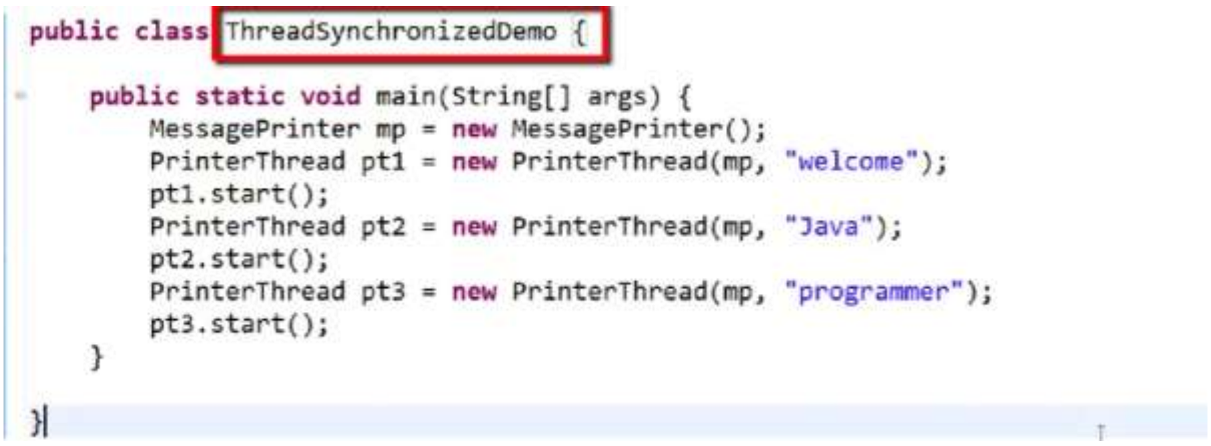


```
package com.demo;

class MessagePrinter {
    public void display(String msg) {
        System.out.print("<" + msg);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println(">");
    }
}
```

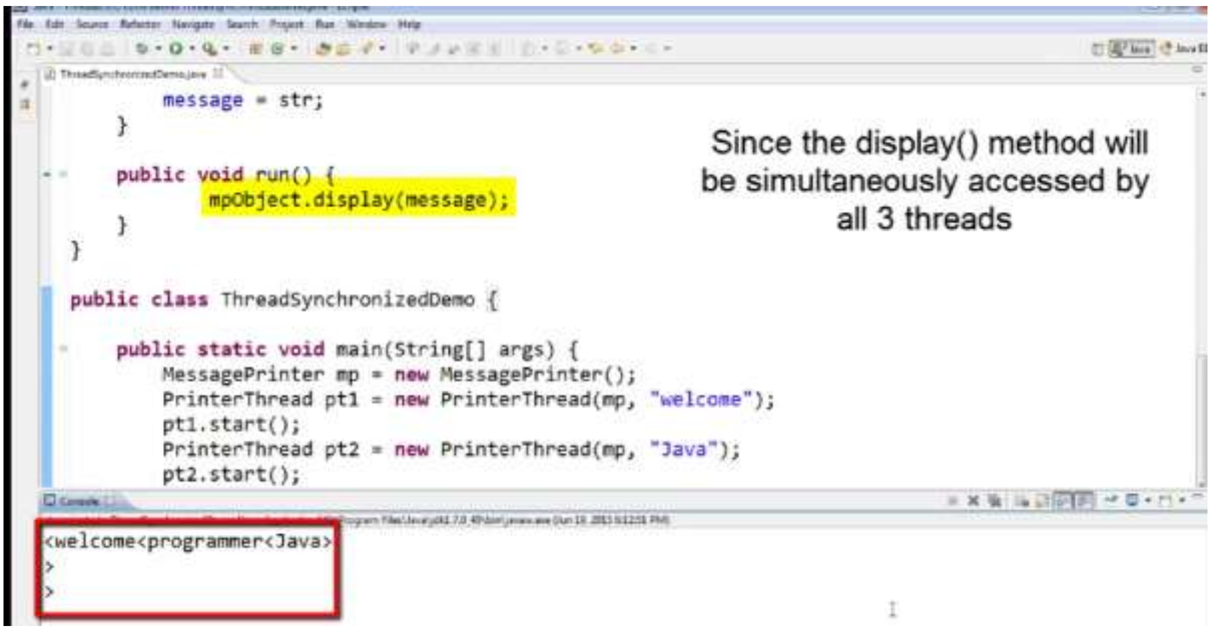
< String value >



```
class PrinterThread extends Thread {

    private String message;
    private MessagePrinter mpObject;

    public PrinterThread(MessagePrinter mp, String str) {
        mpObject = mp;
        message = str;
    }

    public void run() {
            mpObject.display(message);
    }
}
```



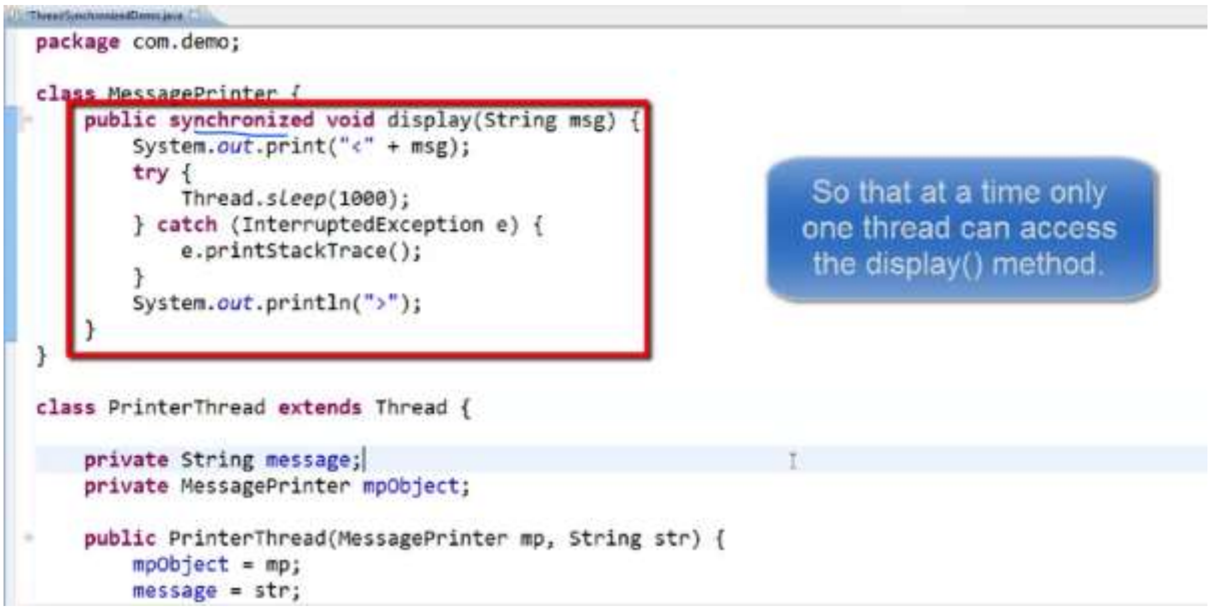
```
public class ThreadSynchronizedDemo {

    public static void main(String[] args) {
        MessagePrinter mp = new MessagePrinter();
        PrinterThread pt1 = new PrinterThread(mp, "welcome");
        pt1.start();
        PrinterThread pt2 = new PrinterThread(mp, "Java");
        pt2.start();
        PrinterThread pt3 = new PrinterThread(mp, "programmer");
        pt3.start();
    }

}
```



```
            message = str;
        }

        public void run() {
            mpObject.display(message);
        }
    }

    public class ThreadSynchronizedDemo {

        public static void main(String[] args) {
            MessagePrinter mp = new MessagePrinter();
            PrinterThread pt1 = new PrinterThread(mp, "welcome");
            pt1.start();
            PrinterThread pt2 = new PrinterThread(mp, "Java");
            pt2.start();
```

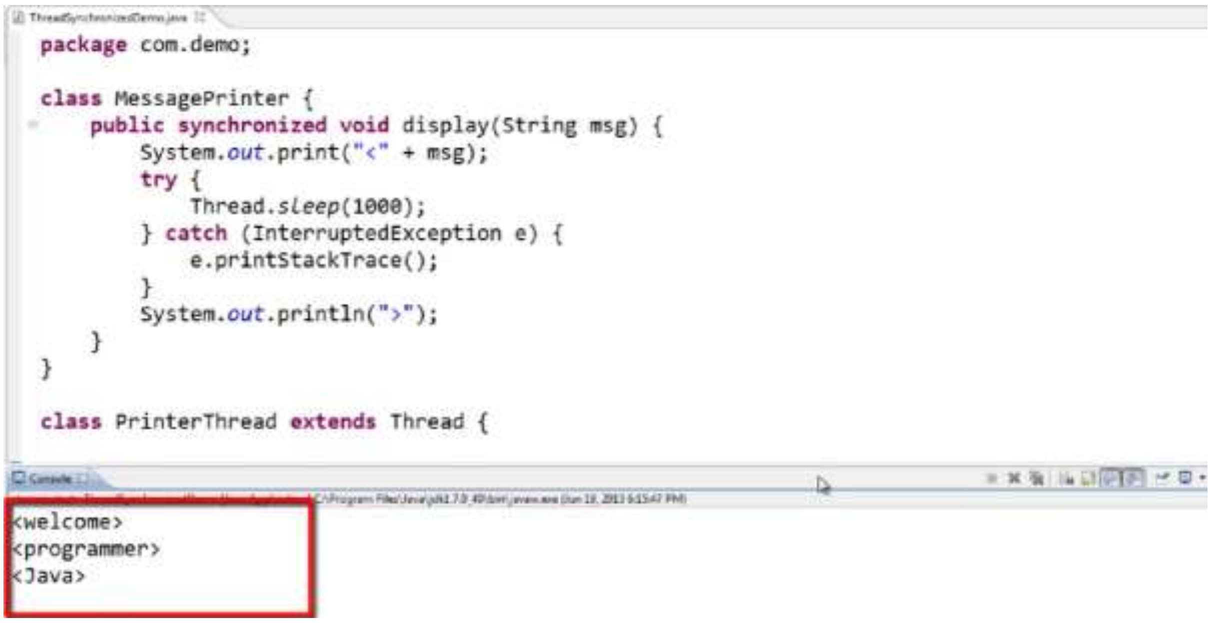
Since the display() method will be simultaneously accessed by all 3 threads

```
<welcome<programmer<Java>
>
>
```



```
package com.demo;

class MessagePrinter {
    public synchronized void display(String msg) {
        System.out.print("<" + msg);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println(">");
    }
}

class PrinterThread extends Thread {

    private String message;
    private MessagePrinter mpObject;

    public PrinterThread(MessagePrinter mp, String str) {
        mpObject = mp;
        message = str;
```

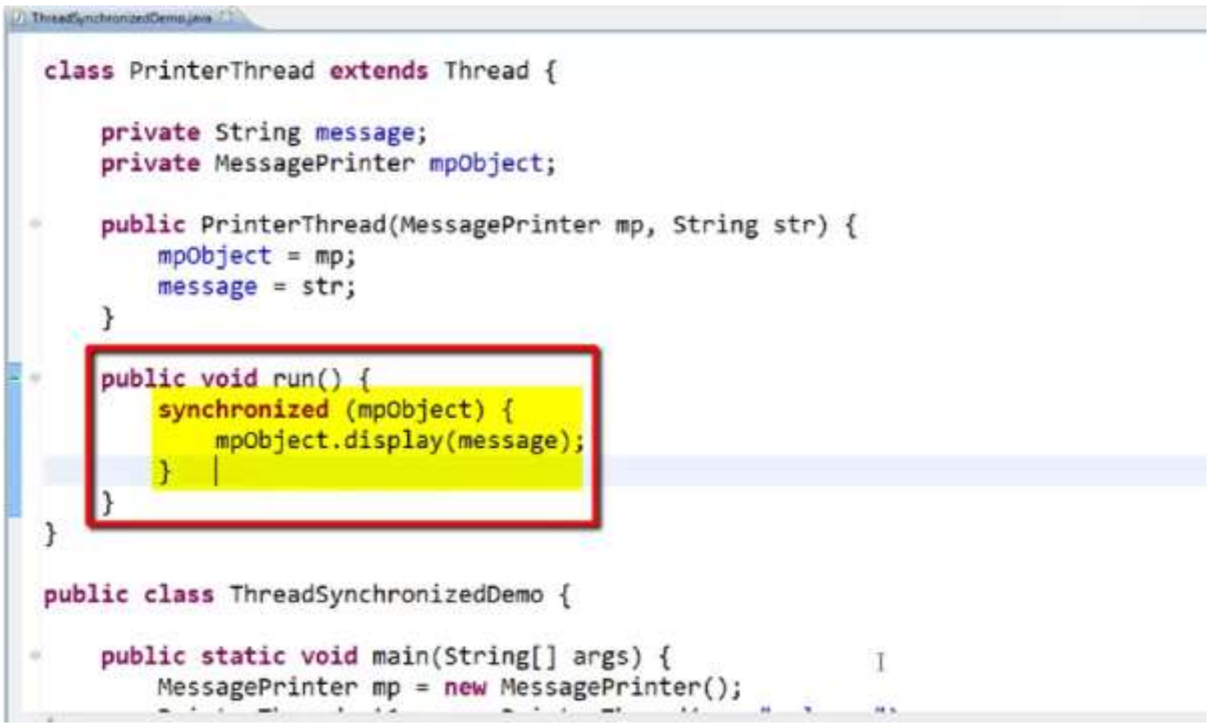
So that at a time only one thread can access the display() method.



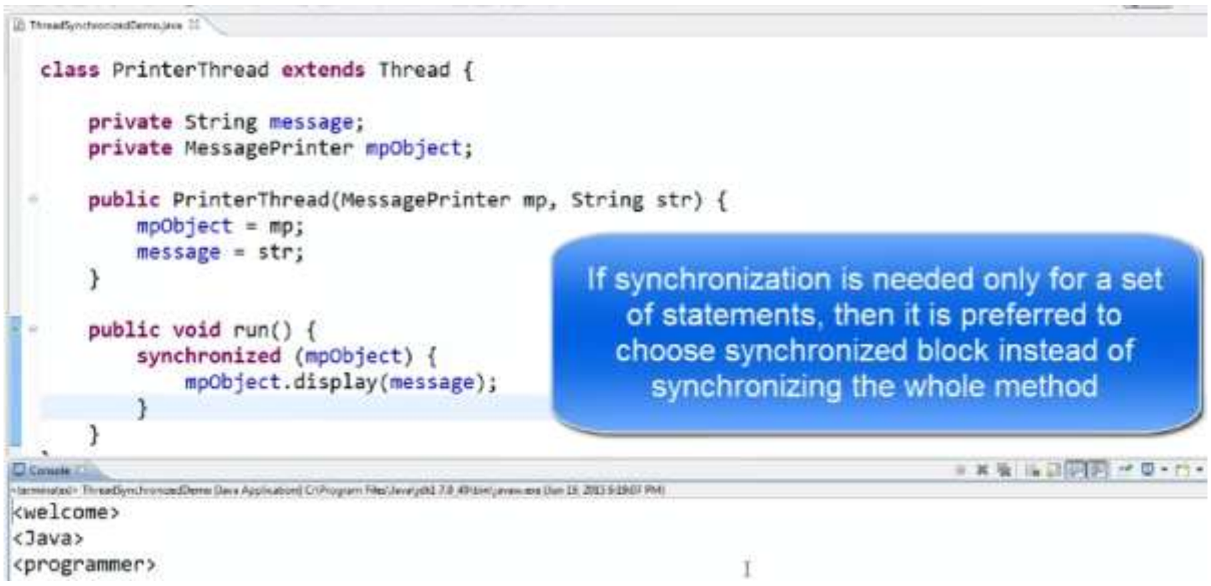
```
package com.demo;

class MessagePrinter {
    public synchronized void display(String msg) {
        System.out.print("<" + msg);
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println(">");
    }
}

class PrinterThread extends Thread {
```

```
<welcome>
<programmer>
<Java>
```

Synchronized can be applied at block level also



```
class PrinterThread extends Thread {

    private String message;
    private MessagePrinter mpObject;

    public PrinterThread(MessagePrinter mp, String str) {
        mpObject = mp;
        message = str;
    }

    public void run() {
        synchronized (mpObject) {
            mpObject.display(message);
        }
    }
}

public class ThreadSynchronizedDemo {

    public static void main(String[] args) {
        MessagePrinter mp = new MessagePrinter();
```



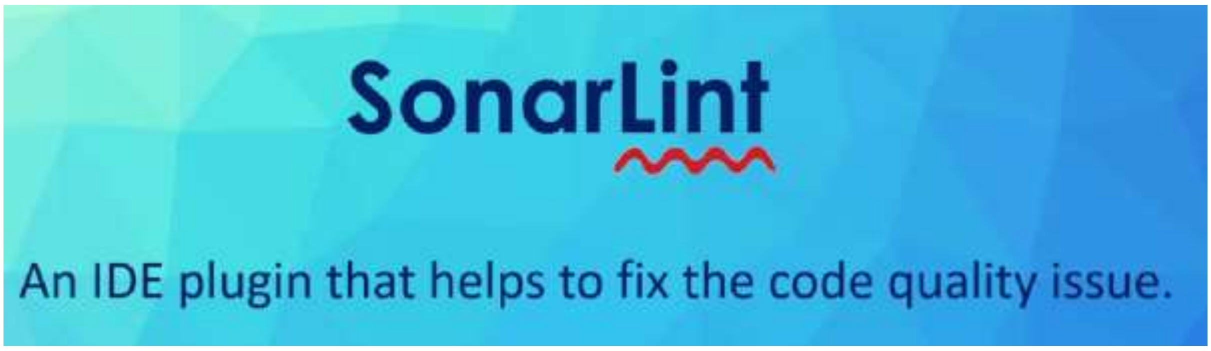
ThreadSynchronizedDemo.java

```
class PrinterThread extends Thread {

    private String message;
    private MessagePrinter mpObject;

    public PrinterThread(MessagePrinter mp, String str) {
        mpObject = mp;
        message = str;
    }

    public void run() {
        synchronized (mpObject) {
            mpObject.display(message);
        }
    }
```

If synchronization is needed only for a set of statements, then it is preferred to choose synchronized block instead of synchronizing the whole method

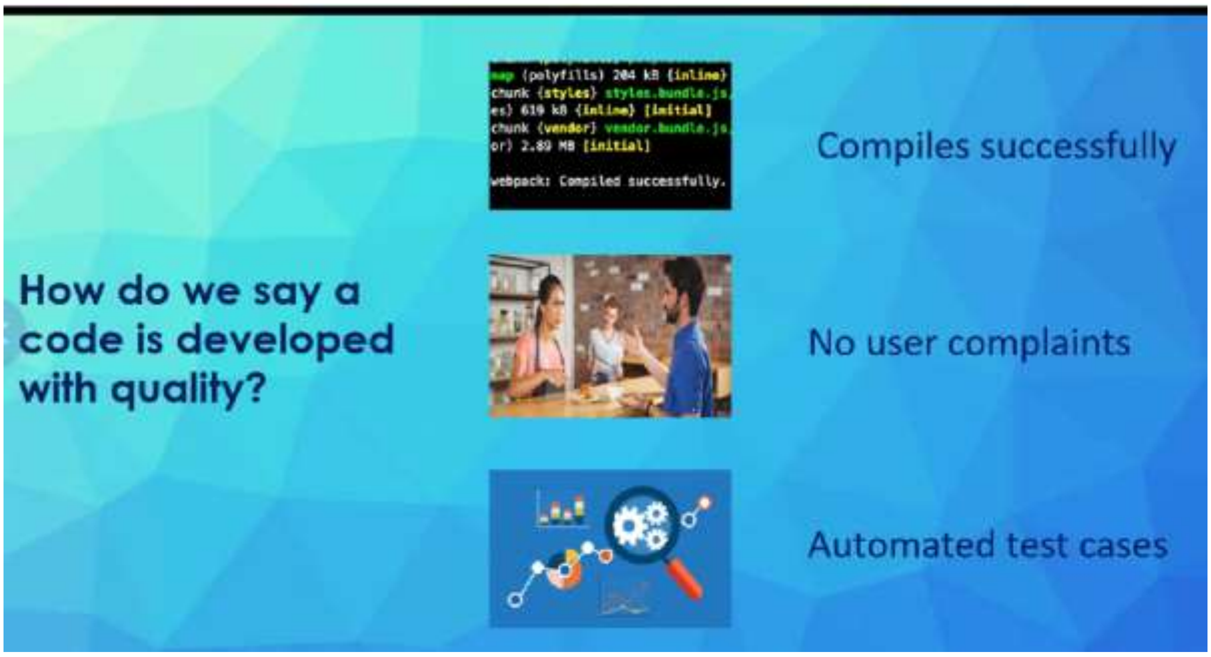
Console

```
<terminated> ThreadSynchronizedDemo [Java Application] C:\Program Files\Java\jdk1.7.0_45\bin\javaw.exe (Jun 15, 2013 5:19:07 PM)
<welcome>
<Java>
<programmer>
```



SonarLint

An IDE plugin that helps to fix the code quality issue.



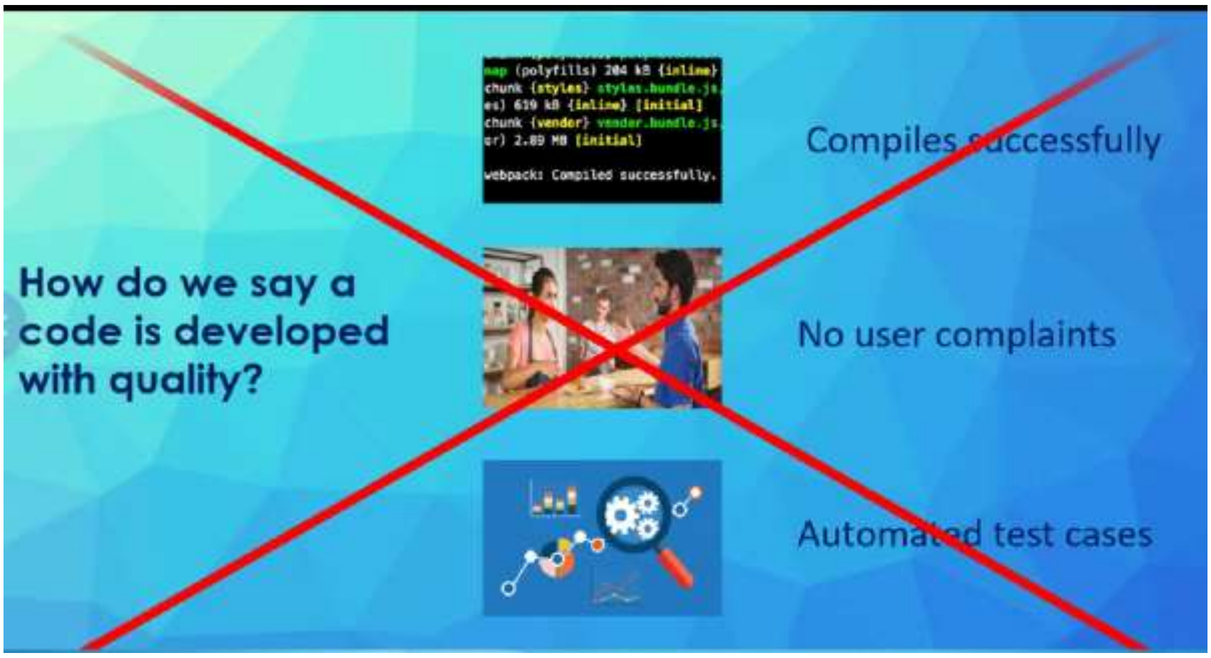
How do we say a code is developed with quality?

```
map (polyfills) 204 kB {inline}
chunk {styles} styles.bundle.js,
es) 619 kB {inline} [initial]
chunk {vendor} vendor.bundle.js
or) 2.89 MB [initial]

webpack: Compiled successfully.
```

Compiles successfully

No user complaints

Automated test cases



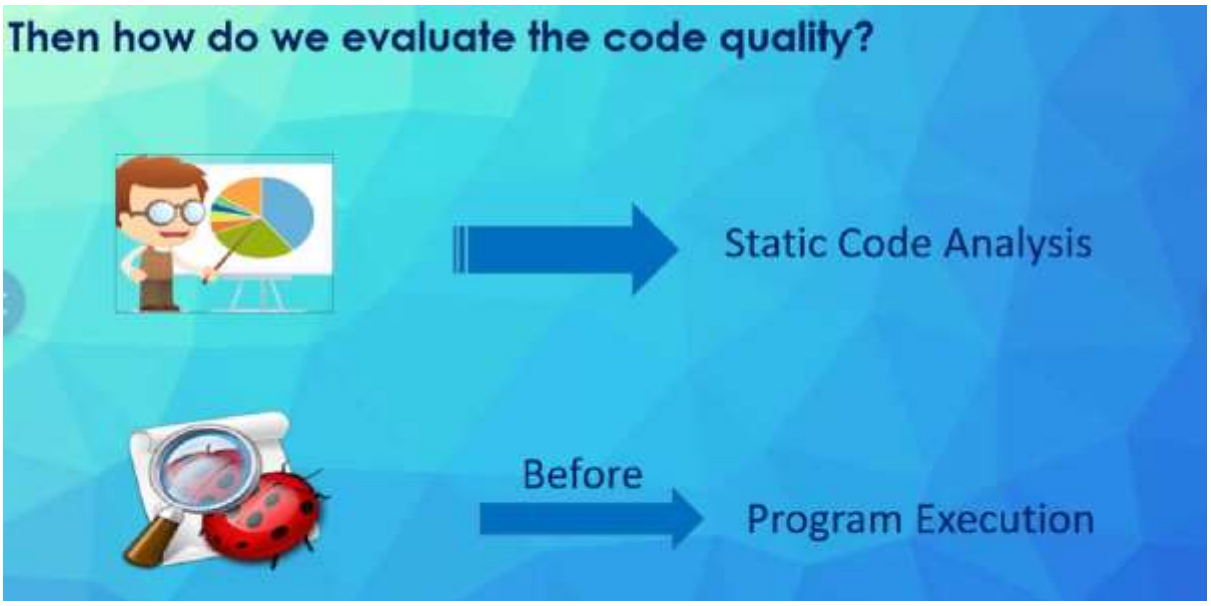
How do we say a code is developed with quality?

```
map (polyfills) 204 kB {inline}
chunk {styles} styles.bundle.js,
es) 619 kB {inline} [initial]
chunk {vendor} vendor.bundle.js,
er) 2.89 MB [initial]

webpack: Compiled successfully.
```

~~Compiles successfully~~

~~No user complaints~~

~~Automated test cases~~

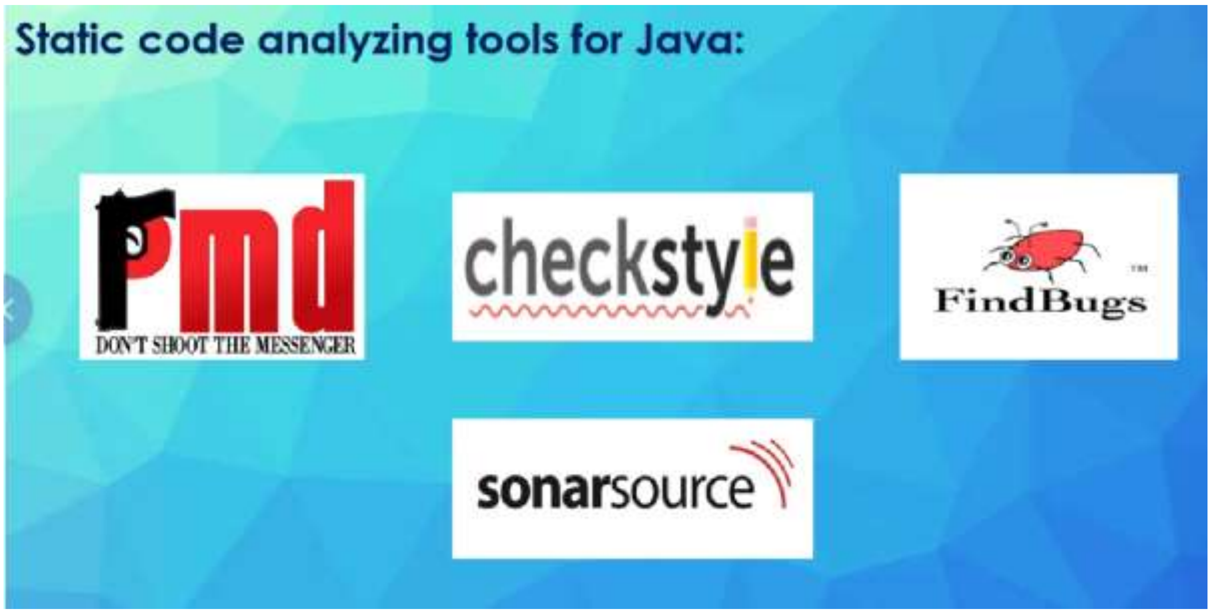


Then how do we evaluate the code quality?

Static Code Analysis

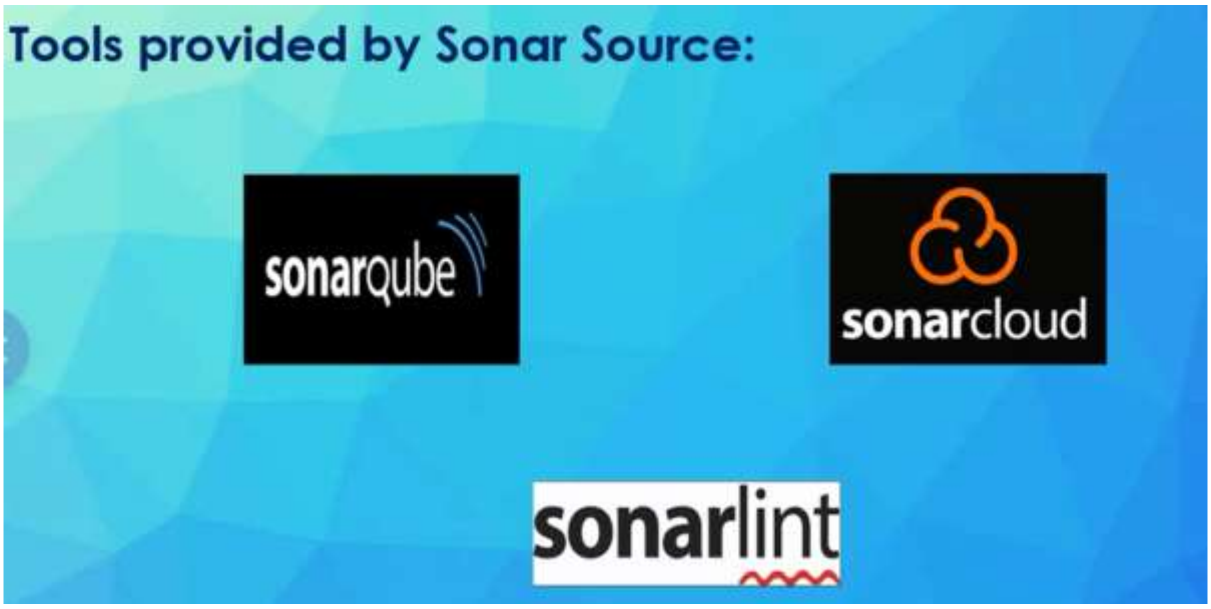
Before

Program Execution



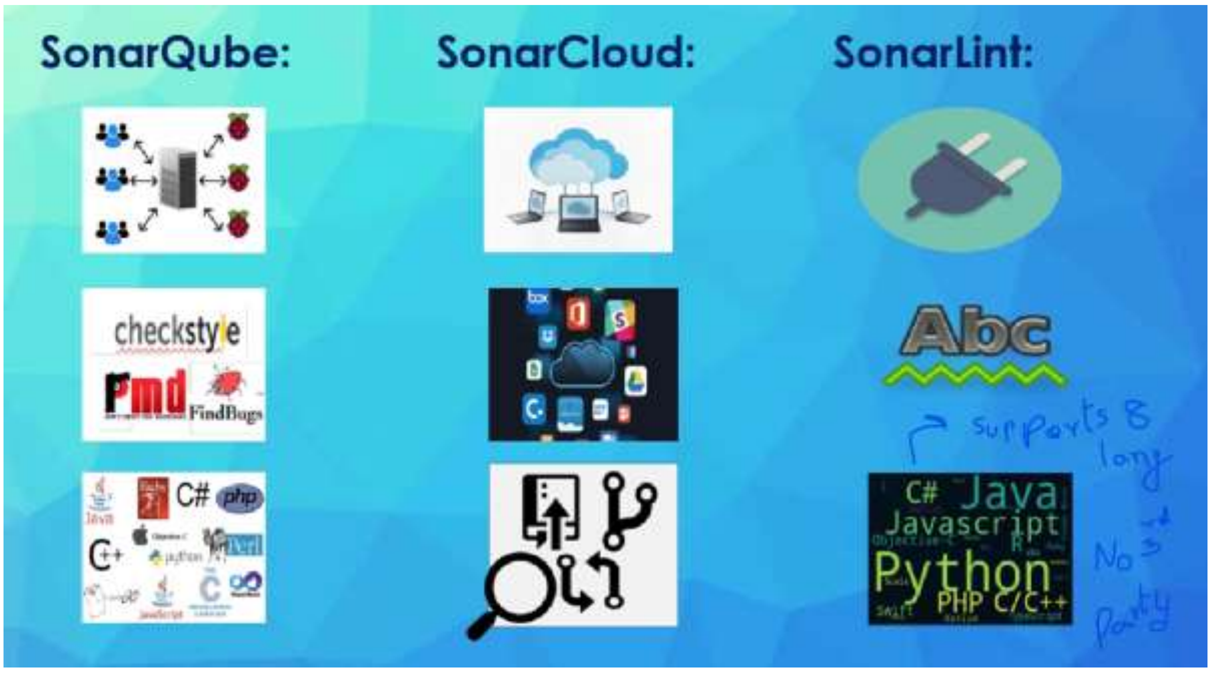
Static code analyzing tools for Java:

Pmd
DON'T SHOOT THE MESSENGER
checkstyle
FindBugs
TM
sonarsource



Tools provided by Sonar Source:

sonarqube
sonarcloud
sonarlint



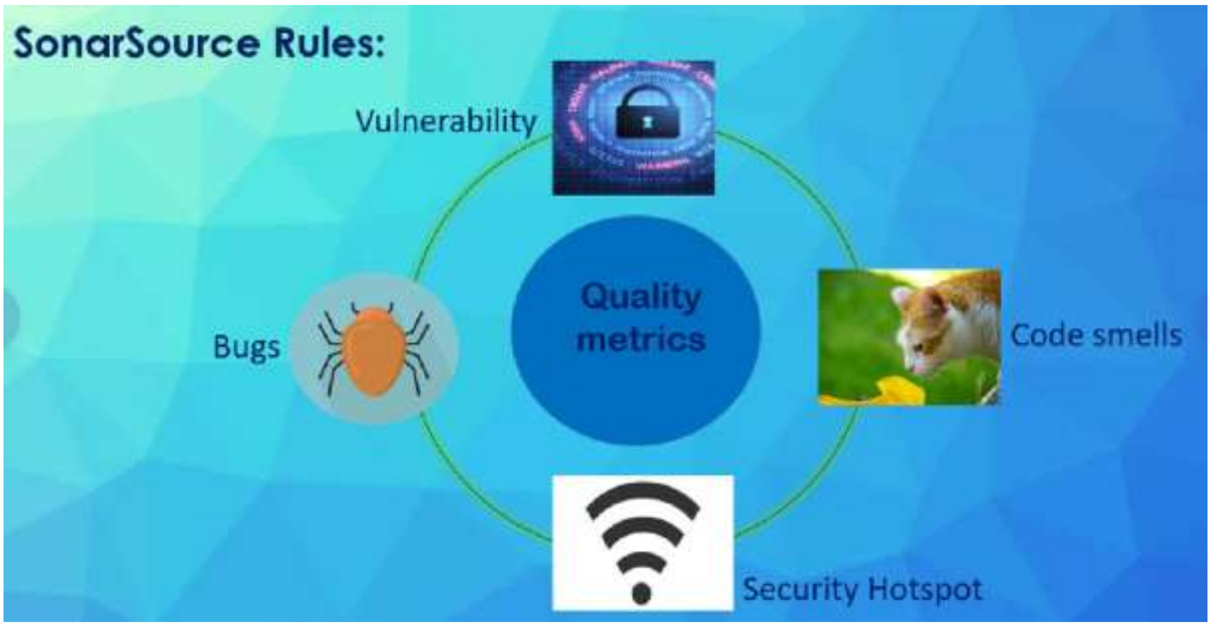
SonarQube:

SonarCloud:

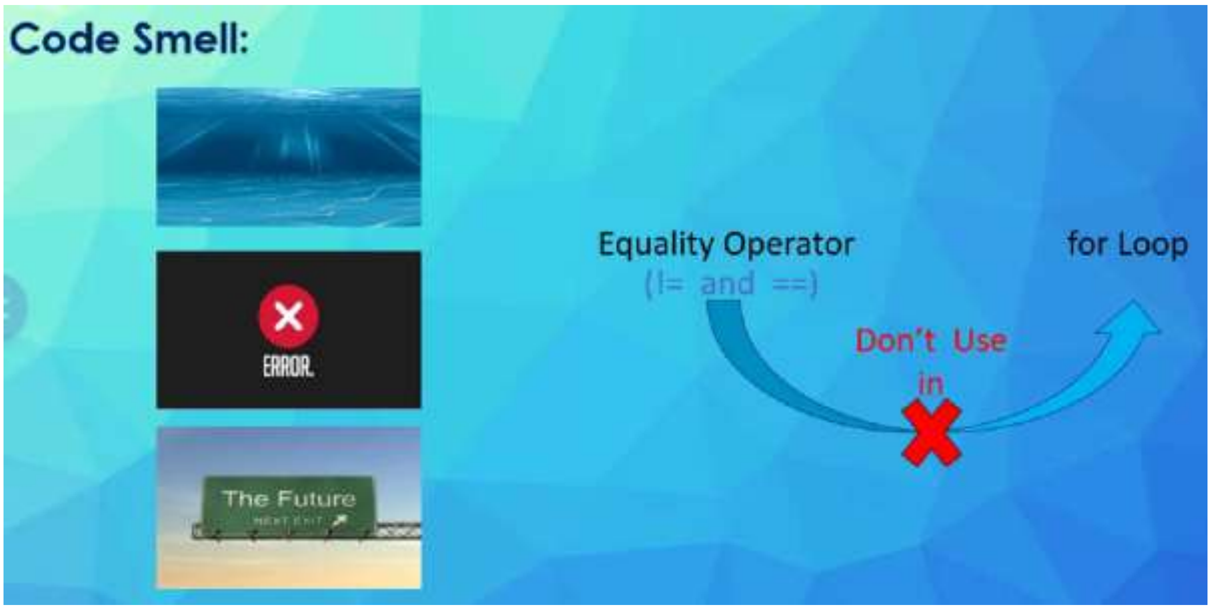
SonarLint:

supports 8 lang

No 3rd party



SonarSource Rules:

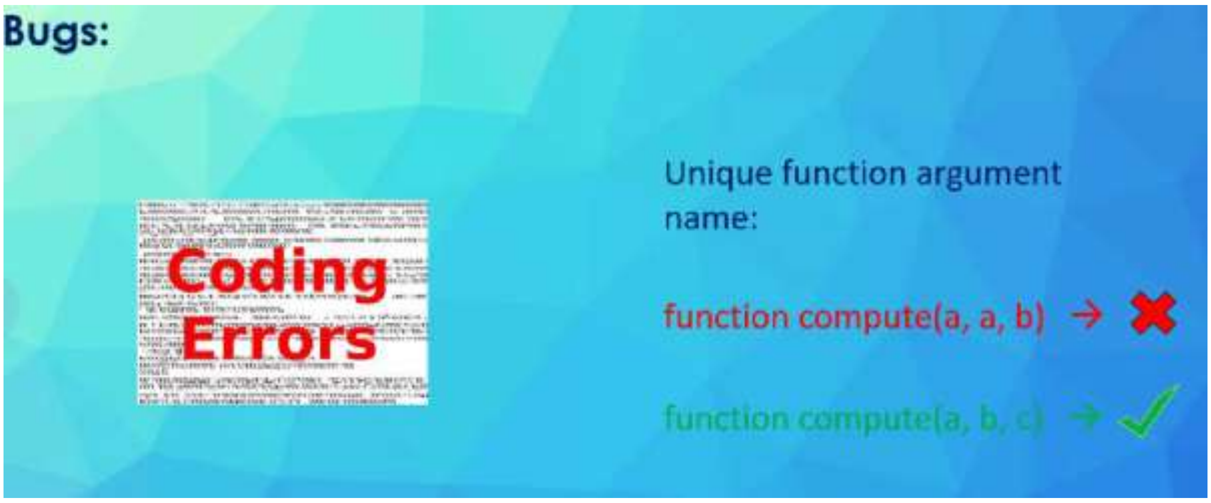

ERROR.
The Future
NEXT EXIT

Code Smell:

Equality Operator
(!= and ==)

Don't Use in

for Loop

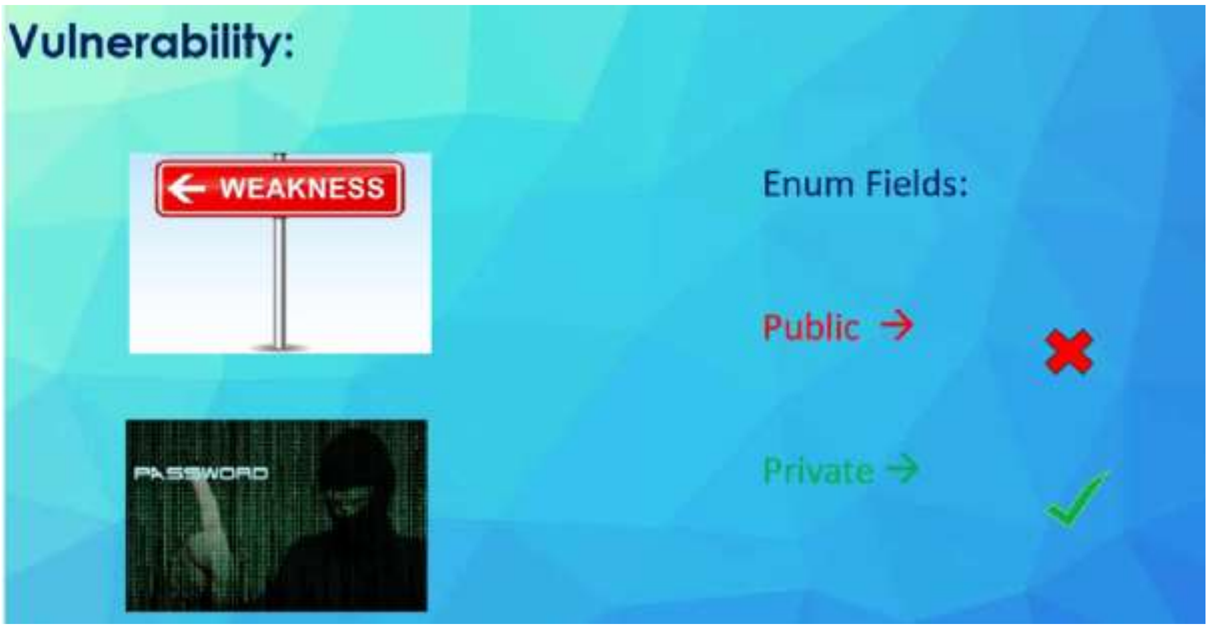

Coding Errors

Bugs:

Unique function argument name:

function compute(a, a, b) → ✗

function compute(a, b, c) → ✓

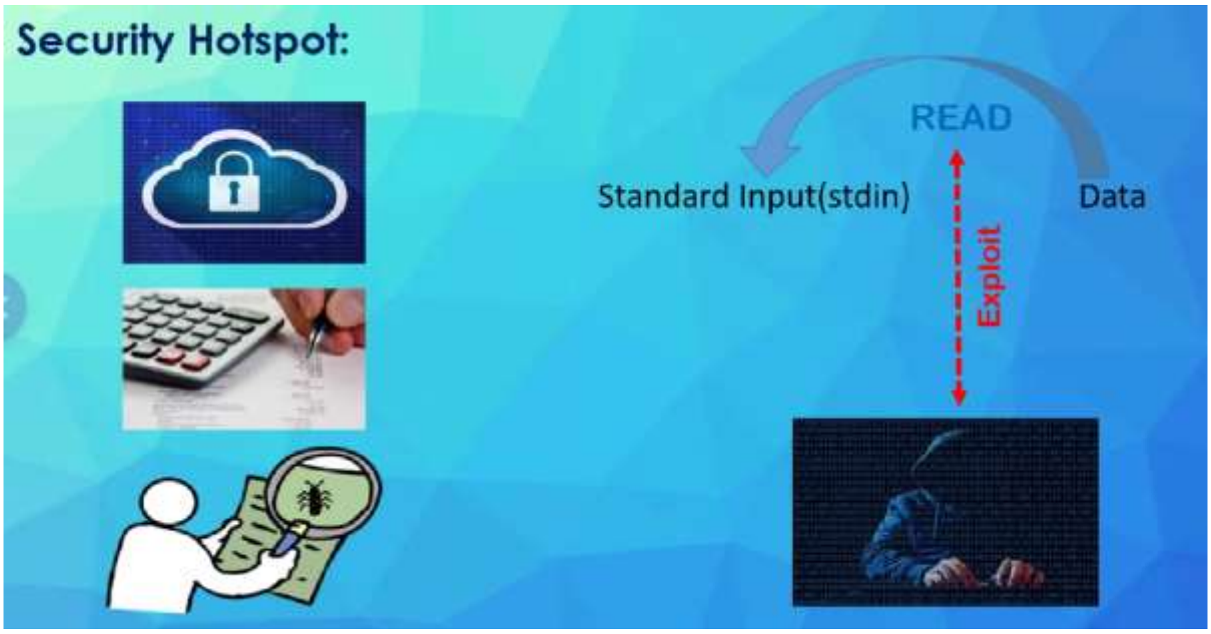

← WEAKNESS
PASSWORD

Vulnerability:

Enum Fields:

Public → ✗

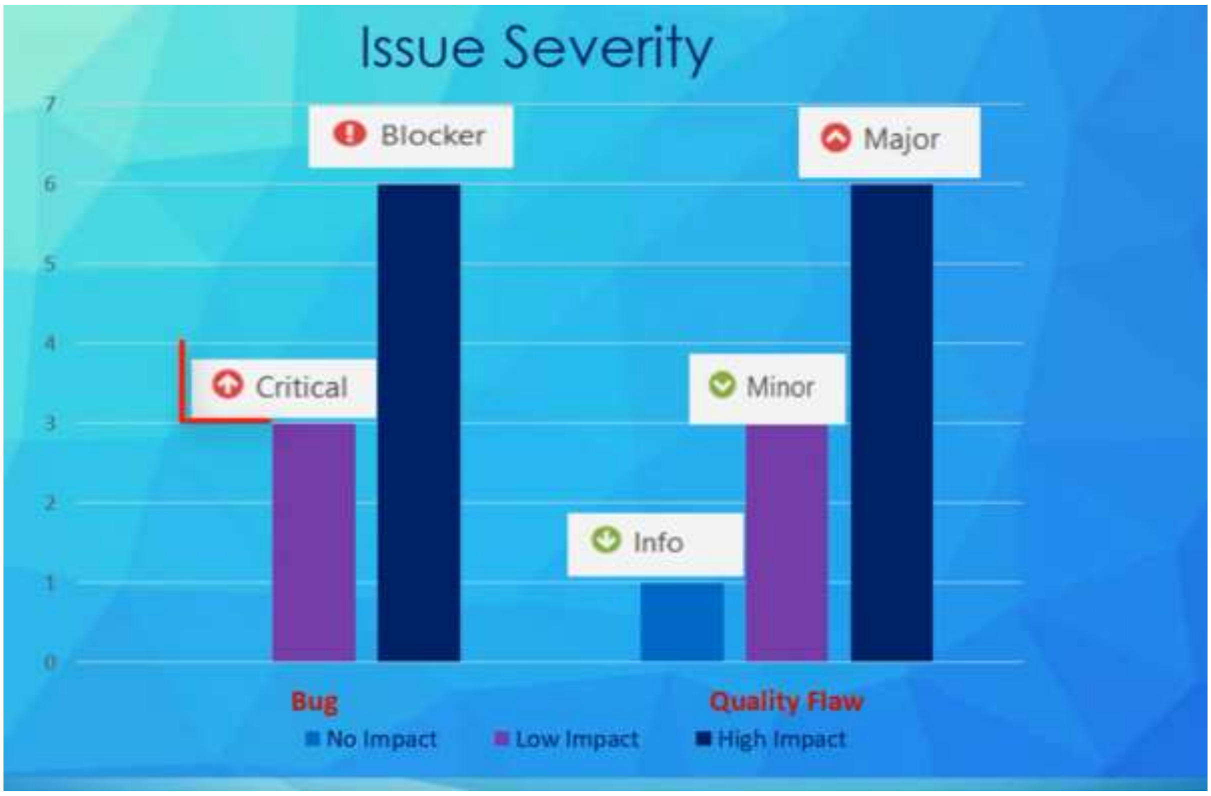
Private → ✓

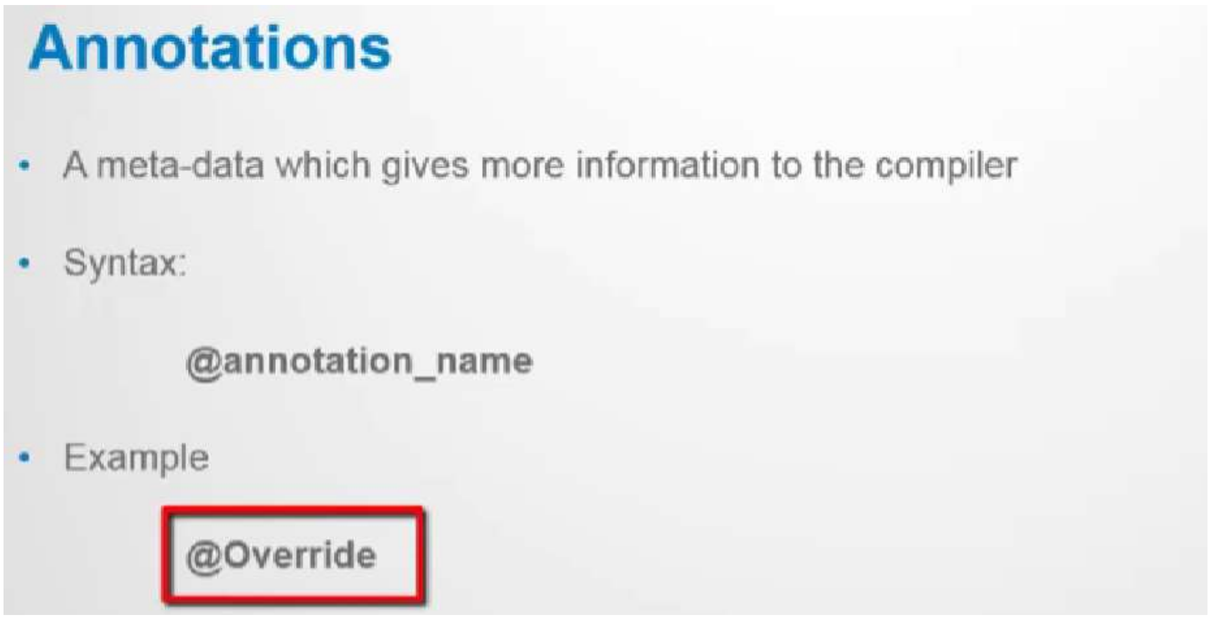


Install sonar lint in eclipse:

Go to eclipse market place → search sonarlint and install

Sonarlint has different views → go to windows – show view – others – sonarlint





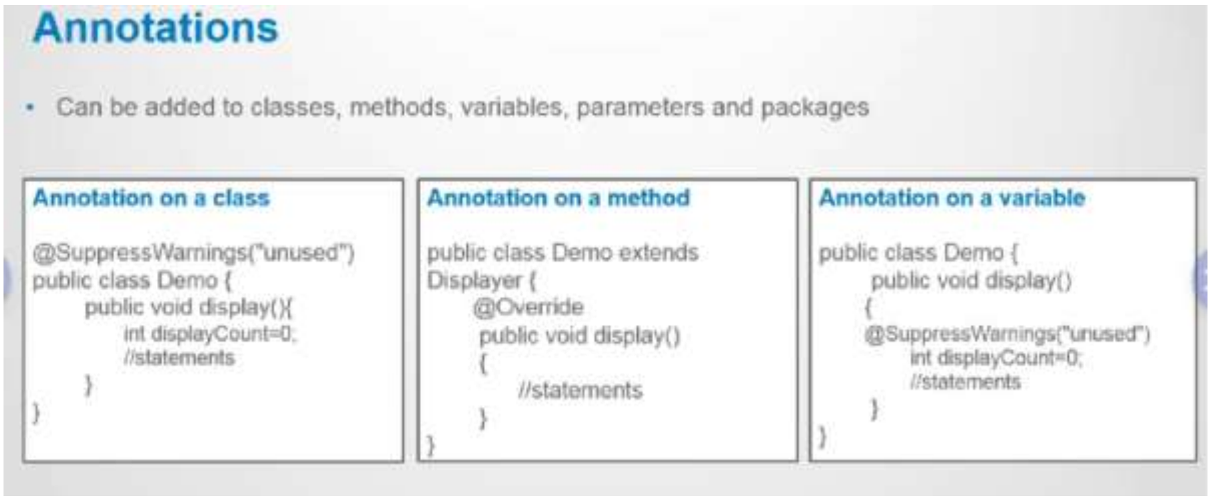
Annotations

- A meta-data which gives more information to the compiler
- Syntax:

  @annotation_name

- Example

  @Override



Annotations

- Can be added to classes, methods, variables, parameters and packages

Annotation on a class

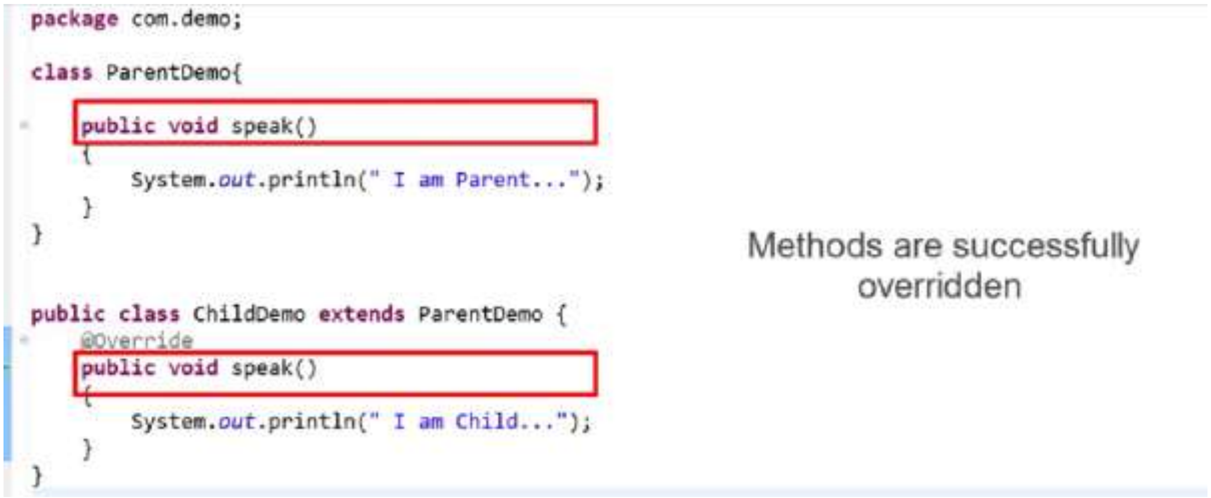
```
@SuppressWarnings("unused")
public class Demo {
      public void display(){
            int displayCount=0;
            //statements
      }
}
```

Annotation on a method

```
public class Demo extends
Displayer {
      @Override
      public void display()
      {
            //statements
      }
}
```

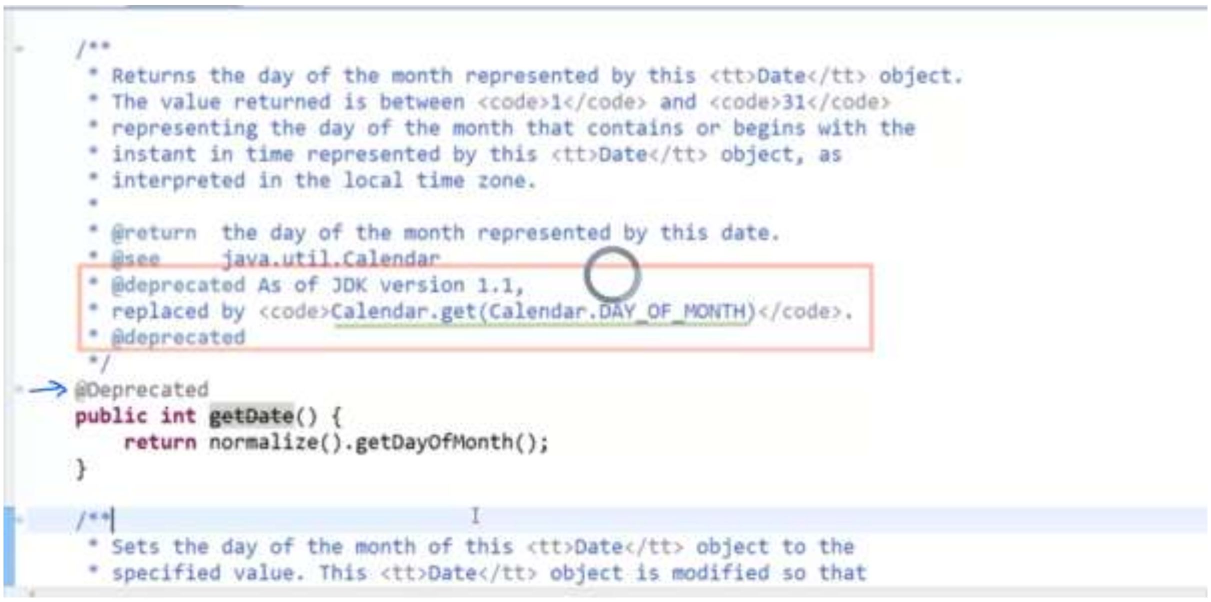
Annotation on a variable

```
public class Demo {
      public void display()
      {
      @SuppressWarnings("unused")
            int displayCount=0;
            //statements
      }
}
```



```
package com.demo;

class ParentDemo{

    public void speak()
    {
        System.out.println(" I am Parent...");
    }
}

public class ChildDemo extends ParentDemo {
    @Override
    public void speak()
    {
        System.out.println(" I am Child...");
    }
}
```

Methods are successfully overridden



```
    /**
     * Returns the day of the month represented by this <tt>Date</tt> object.
     * The value returned is between <code>1</code> and <code>31</code>
     * representing the day of the month that contains or begins with the
     * instant in time represented by this <tt>Date</tt> object, as
     * interpreted in the local time zone.
     *
     * @return  the day of the month represented by this date.
     * @see     java.util.Calendar
     * @deprecated As of JDK version 1.1,
     * replaced by <code>Calendar.get(Calendar.DAY_OF_MONTH)</code>.
     * @deprecated
     */
    @Deprecated
    public int getDate() {
        return normalize().getDayOfMonth();
    }

    /**
     * Sets the day of the month of this <tt>Date</tt> object to the
     * specified value. This <tt>Date</tt> object is modified so that
```

Garbage Collector

- **Memory leaks**
- In programming languages like C , it is responsibility of the programmer to de-allocate memory
- However in Java , Garbage Collector solves the burden of freeing the unused memory automatically. This technique is called **Garbage Collection**
- The objects which are not being referenced can be thrown away
- Garbage Collector determines the objects which are not being referenced by the program and free that memory area occupied by such unreferenced objects
- System.gc() can be used to initiate garbage collection programmatically , but it is not guaranteed because JVM may not run the Garbage Collector immediately

Memory allocation and Garbage Collection

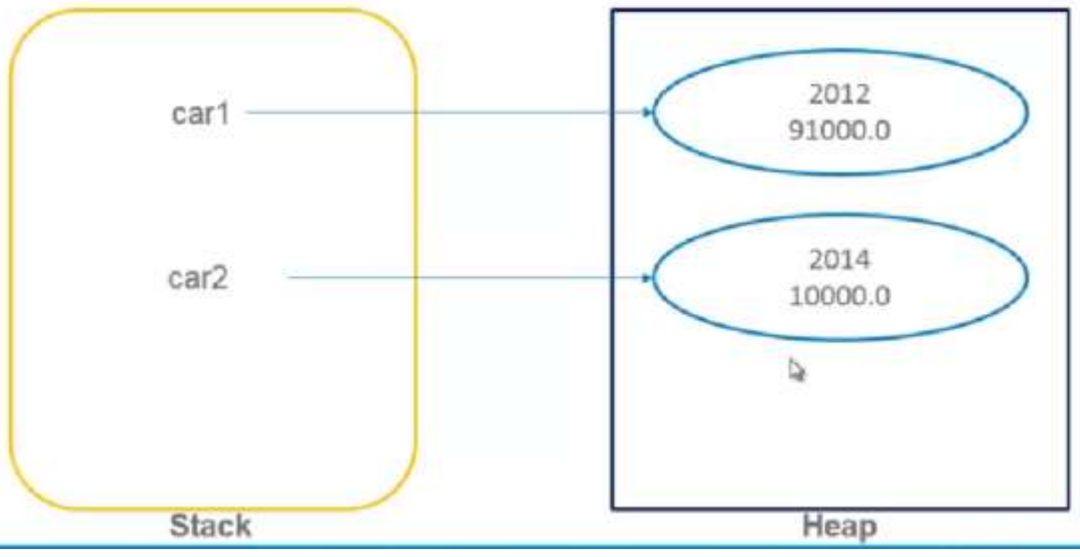
- All local variables are stored into the stack
- All objects are stored in the heap
- An object becomes eligible for Garbage Collection :
  - When the object is not being referenced by any reference
  - When the program is terminated

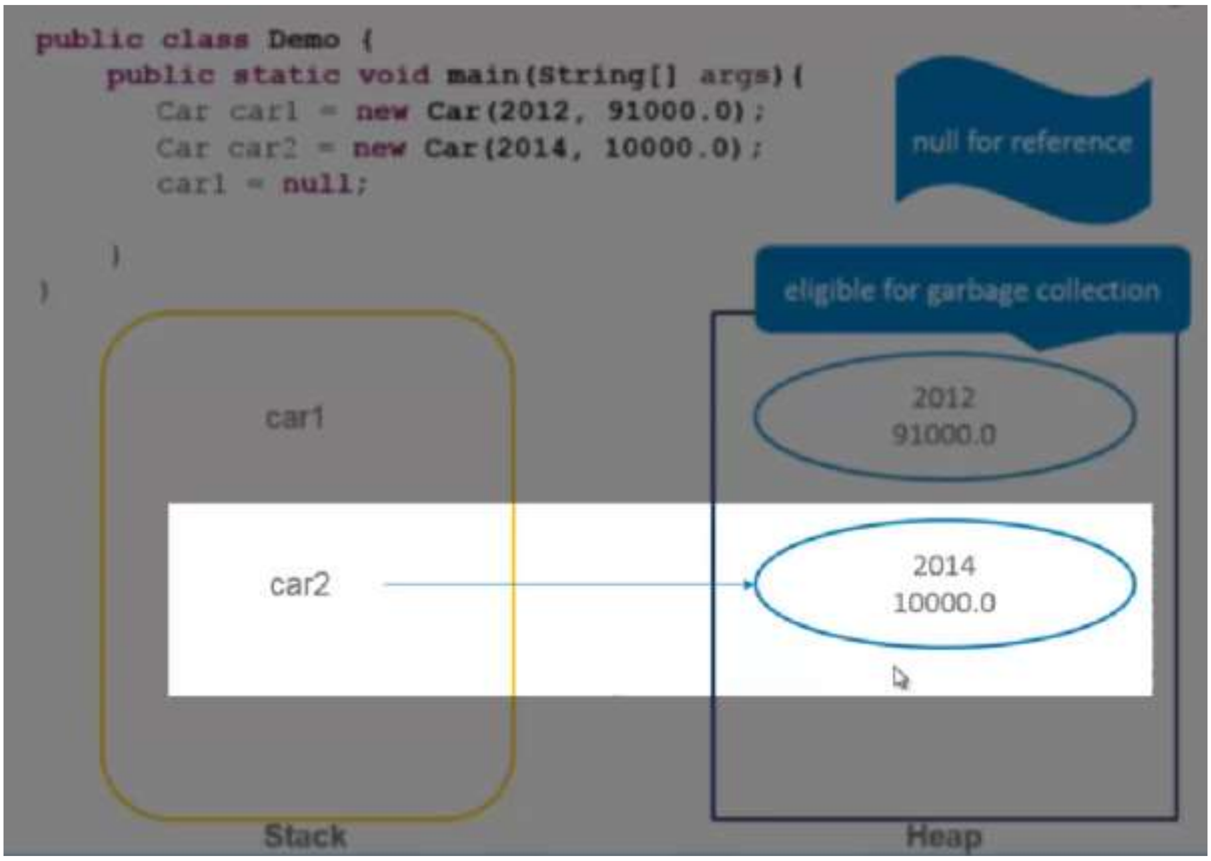
Garbage Collector

- Lets take this Car class as an example.

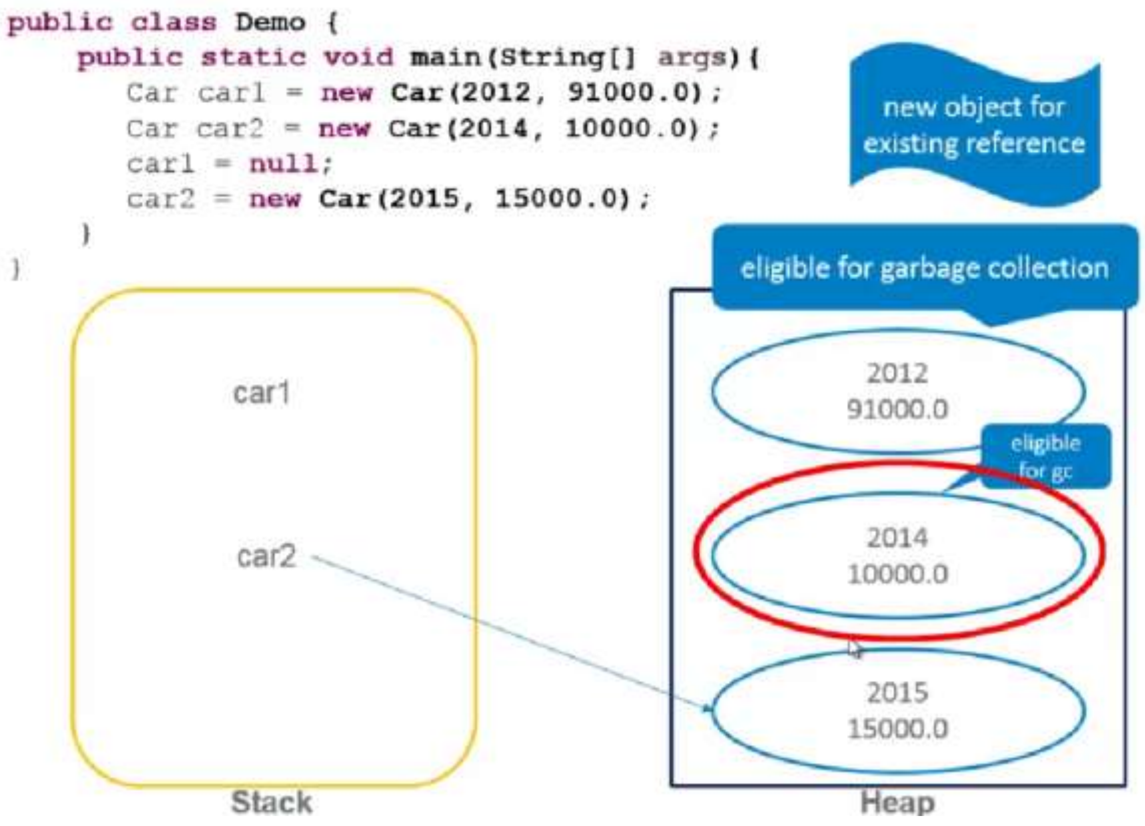
```
class Car {
   private int carModel;
   private double price;

   public Car(int carModel, double price){
       this.carModel = carModel;
       this.price = price;
   }
}
public class Demo {
    public static void main(String[] args){

    }
}
```

```
public class Demo {
    public static void main(String[] args){
       Car car1 = new Car(2012, 91000.0);
       Car car2 = new Car(2014, 10000.0);
    }
}
```





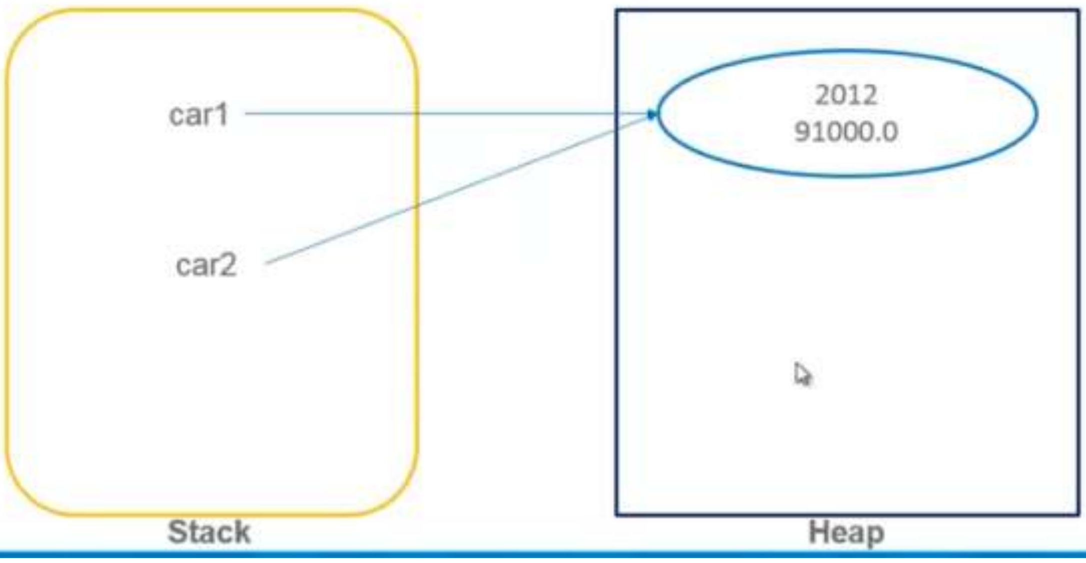
```
public class Demo {
    public static void main(String[] args){
       Car car1 = new Car(2012, 91000.0);
       Car car2 = new Car(2014, 10000.0);
       car1 = null;

    }
}
```



```
public class Demo {
    public static void main(String[] args){
       Car car1 = new Car(2012, 91000.0);
       Car car2 = new Car(2014, 10000.0);
       car1 = null;
       car2 = new Car(2015, 15000.0);
    }
}
```

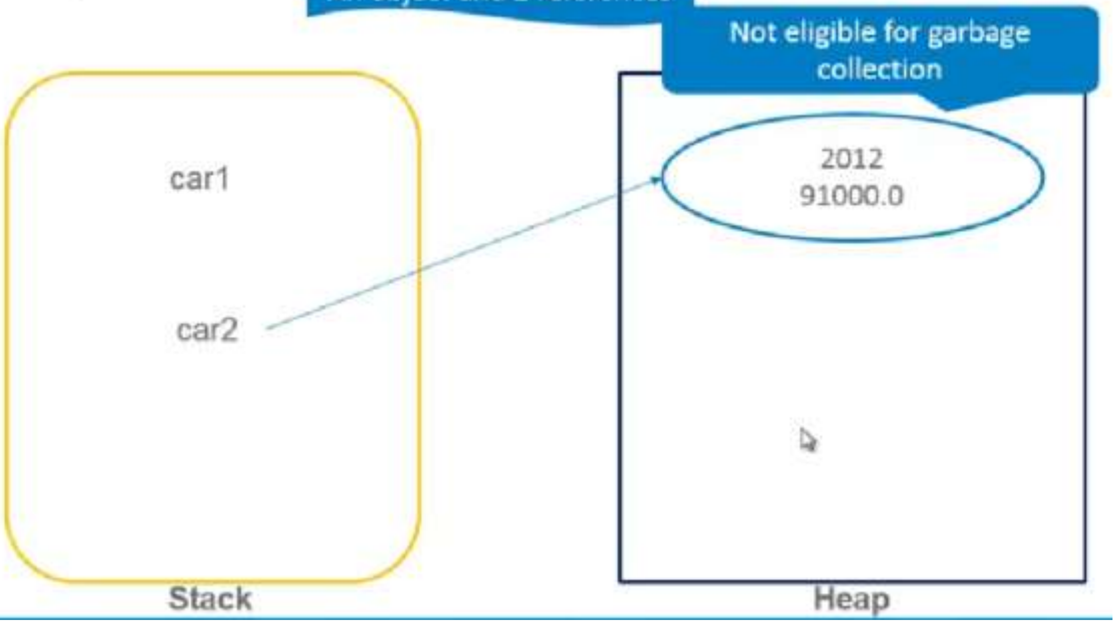
```
public class Demo {
    public static void main(String[] args){
        Car car1 = new Car(2012, 91000.0);
        Car car2= car1;

        }
}
```

An object and 2 references



```
public class Demo {
    public static void main(String[] args){
        Car car1 = new Car(2012, 91000.0);
        Car car2= car1;
        car1 = null;
        }
}
```

An object and 2 references



Summary

- Memory leaks
- Memory allocation
  - stack
  - heap
- Garbage Collection
  - object not being referenced by any reference
  - program is terminated