SQL Injection

SQL Injection is one of the many vulnerabilities in the web applications. SQL Injection as the name suggests is adding or inserting user input data into a database using query. Through SQL injection a malicious user can obtain unauthorized access to the application and steal data. There are tools like Postman which is used to perform these actions. The best solution to counter this issue is by using Prepared Statement in place of Statement while executing the query. Tools like SQLMap are also used to avoid the SQL Injection.

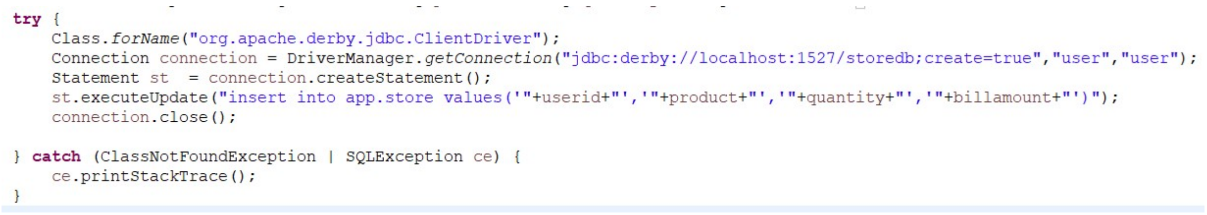
One of the ways to perform the attack is given below.

select name from app.employee where empId=3 or 1=1

DIFFERENT WAYS TO EXECUTE SQL QUERY

There are 3 ways to do this process:

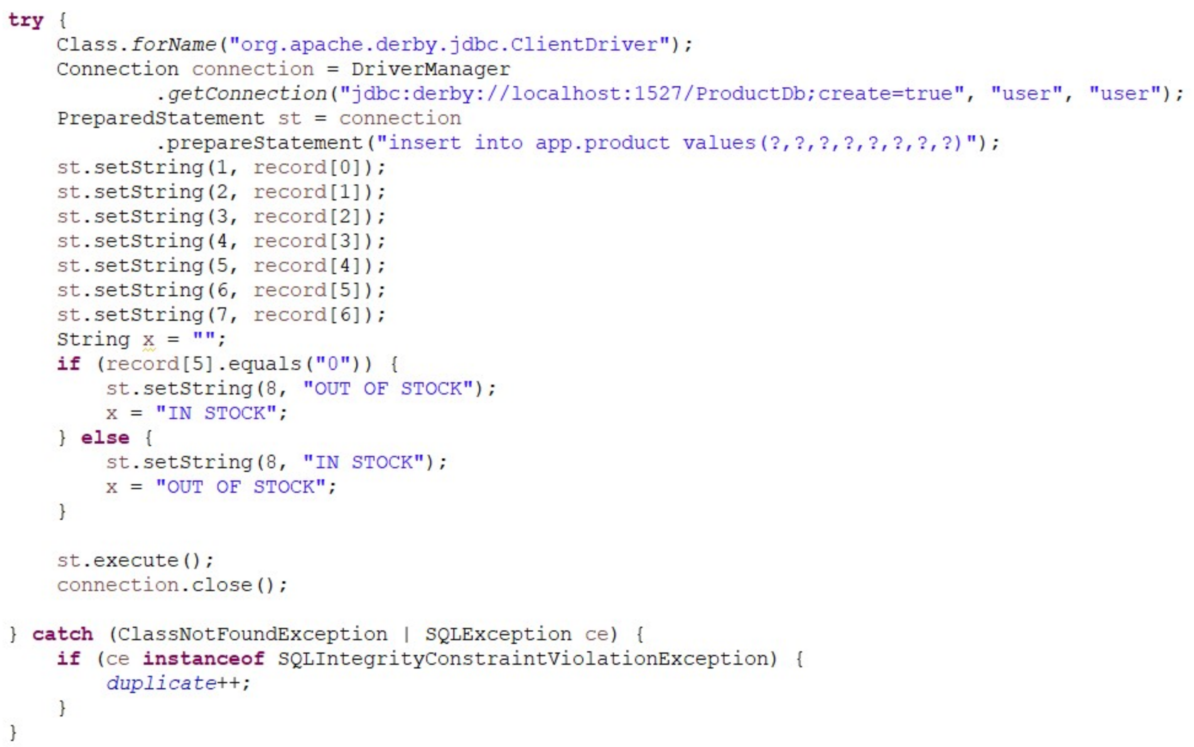
1. Statement: Statement is an interface which is used to execute normal SQL Query. When it is created it's not necessary to provide a query.
   Ex: Statement st = con.createStatement();



```
try {
    Class.forName("org.apache.derby.jdbc.ClientDriver");
    Connection connection = DriverManager.getConnection("jdbc:derby://localhost:1527/storedb;create=true","user","user");
    Statement st  = connection.createStatement();
    st.executeUpdate("insert into app.store values('"+userid+"','"+product+"','"+quantity+"','"+billamount+"')");
    connection.close();

} catch (ClassNotFoundException | SQLException ce) {
    ce.printStackTrace();
}
```

2. Prepared Statement: It's an interface which is used to execute dynamic and parameterised SQL Query. Prepared Statement is quicker than Statement. In case of Statement Query it will be compiled and executed every time unlike Prepared Statement Query which will not be compiled every time but executed. Biggest advantage of Prepared Statement is that it prevents SQL injection.
   Ex: String sqlquery = "insert into employee values(? ? ? ?)";
   preparedStatement pst = con.preparedStatement(sqlquery);

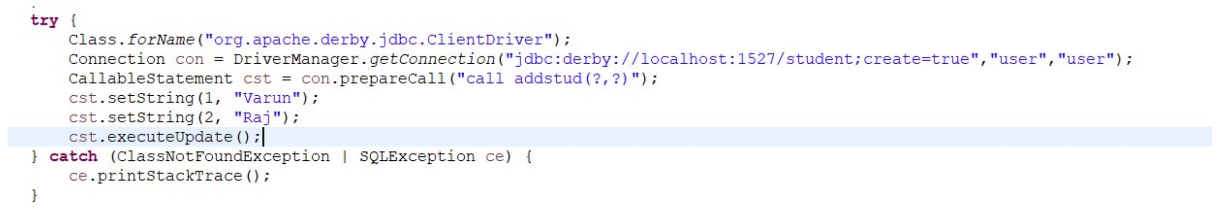


```java
try {
    Class.forName("org.apache.derby.jdbc.ClientDriver");
    Connection connection = DriverManager
            .getConnection("jdbc:derby://localhost:1527/ProductDb;create=true", "user", "user");
    PreparedStatement st = connection
            .prepareStatement("insert into app.product values(?,?,?,?,?,?,?,?)");
    st.setString(1, record[0]);
    st.setString(2, record[1]);
    st.setString(3, record[2]);
    st.setString(4, record[3]);
    st.setString(5, record[4]);
    st.setString(6, record[5]);
    st.setString(7, record[6]);
    String x = "";
    if (record[5].equals("0")) {
        st.setString(8, "OUT OF STOCK");
        x = "IN STOCK";
    } else {
        st.setString(8, "IN STOCK");
        x = "OUT OF STOCK";
    }

    st.execute();
    connection.close();

} catch (ClassNotFoundException | SQLException ce) {
    if (ce instanceof SQLIntegrityConstraintViolationException) {
        duplicate++;
    }
}
```

3. Callable Statement: It is Child Interface of Prepared Statement. Callable Statement is used to execute the Stored Procedure and functions. Similar to method stored procedure has its own parameters. Store Parameters has 3 types of parameters.
    a. IN parameter: IN parameter is used to provide input values.
    b. OUT parameter: OUT parameter is used to collect output values.
    c. INOUT parameter: It is used to provide input and to collect output values.



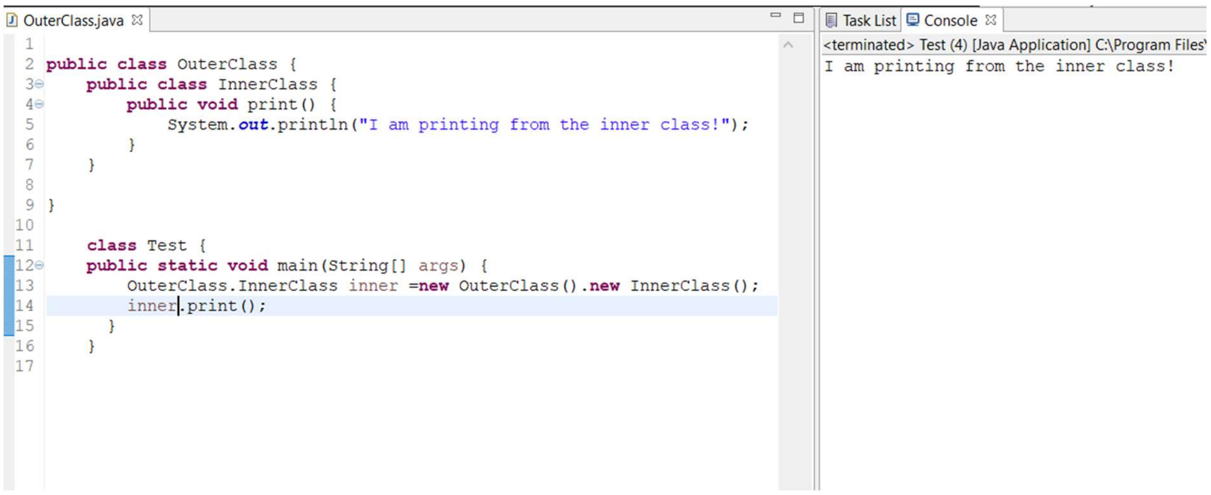
```java
try {
    Class.forName("org.apache.derby.jdbc.ClientDriver");
    Connection con = DriverManager.getConnection("jdbc:derby://localhost:1527/student;create=true","user","user");
    CallableStatement cst = con.prepareCall("call addstud(?,?)");
    cst.setString(1, "Varun");
    cst.setString(2, "Raj");
    cst.executeUpdate();
} catch (ClassNotFoundException | SQLException ce) {
    ce.printStackTrace();
}
```

Inner Classes

Inner classes in Java are the classed which are defined in the scope of another class. They are also called nested classes. Helps in better documentation and maintenance.

There are 4 types of Inner Classes:

1. Nested Inner class: This type of inner class involves nesting of one class inside other. Also, the inner class is capable of accessing the outer class private variables. The inner class access can be modified using access modifiers like private, public, protected and default.



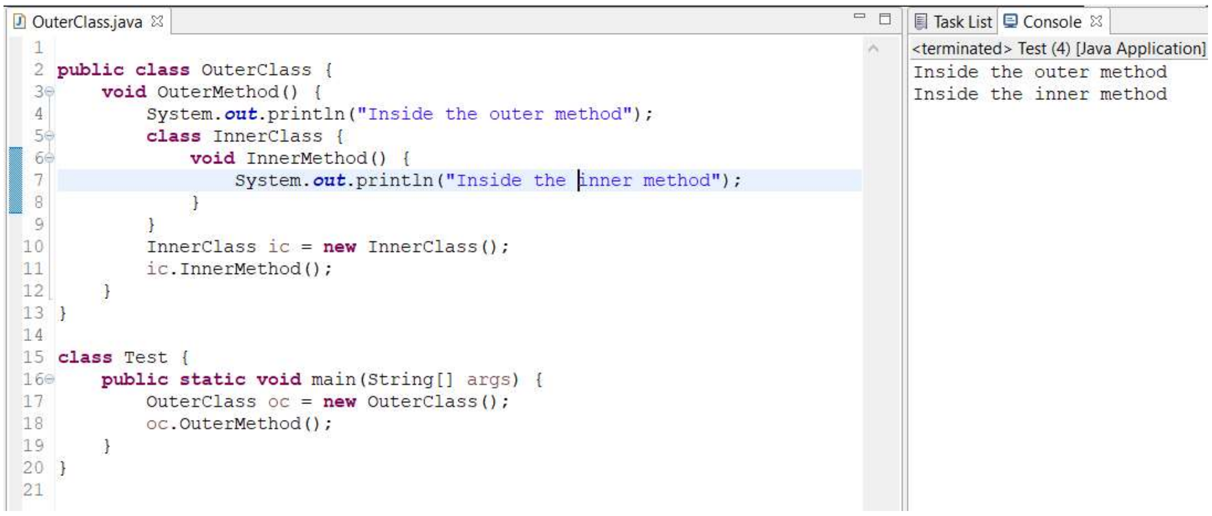
```
public class OuterClass {
    public class InnerClass {
        public void print() {
            System.out.println("I am printing from the inner class!");
        }
    }

}

    class Test {
    public static void main(String[] args) {
        OuterClass.InnerClass inner =new OuterClass().new InnerClass();
        inner.print();
      }
    }
```

```
<terminated> Test (4) [Java Application] C:\Program Files\
I am printing from the inner class!
```

2. Method local inner classes: In the method local inner classes, the outer class methods contains the inner class. Non-final variables created in Outer class couldn’t be accessed in the inner class in JDK 1.7 and below versions but it now it’s available.



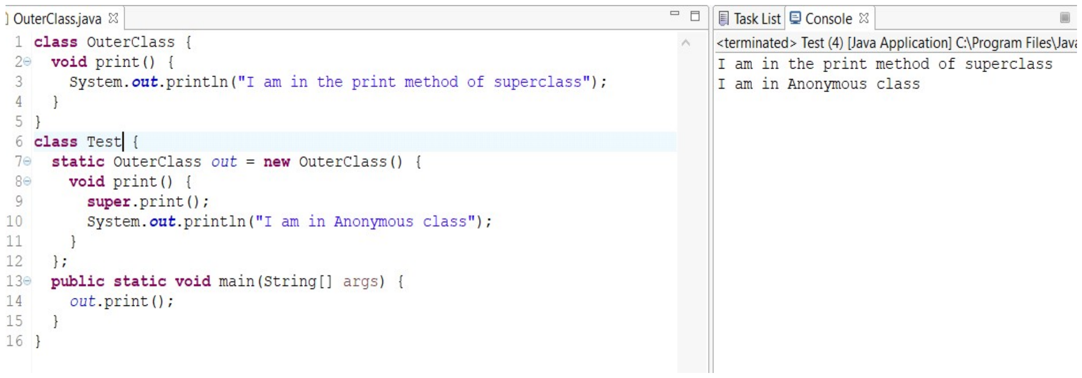
```
public class OuterClass {
    void OuterMethod() {
        System.out.println("Inside the outer method");
        class InnerClass {
            void InnerMethod() {
                System.out.println("Inside the inner method");
            }
        }
        InnerClass ic = new InnerClass();
        ic.InnerMethod();
    }
}

class Test {
    public static void main(String[] args) {
        OuterClass oc = new OuterClass();
        oc.OuterMethod();
    }
}
```

```
<terminated> Test (4) [Java Application]
Inside the outer method
Inside the inner method
```

3. Anonymous inner classes: These inner classes have no name. The definition of these classes is written outside the scope of outer class. There are two types:

a. Subclass of the specified type: The anonymous class (Test) is put in the subclass of the outer class.



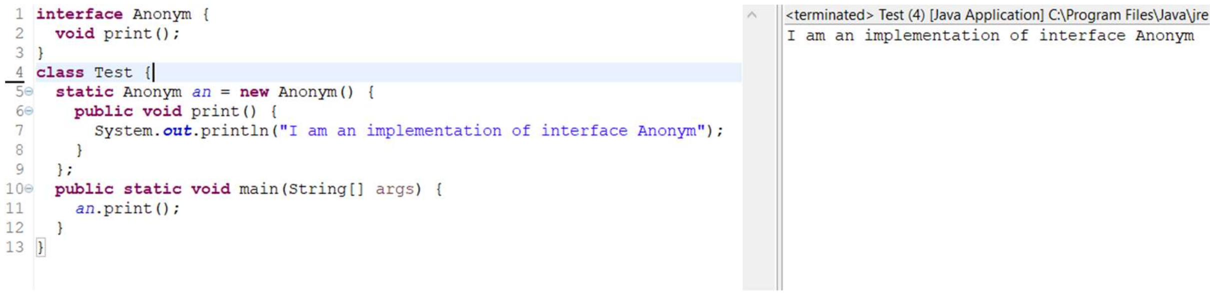
OuterClass.java

```java
class OuterClass {
  void print() {
    System.out.println("I am in the print method of superclass");
  }
}
class Test {
  static OuterClass out = new OuterClass() {
    void print() {
      super.print();
      System.out.println("I am in Anonymous class");
    }
  };
  public static void main(String[] args) {
    out.print();
  }
}
```

Task List | Console

```
<terminated> Test (4) [Java Application] C:\Program Files\Java
I am in the print method of superclass
I am in Anonymous class
```

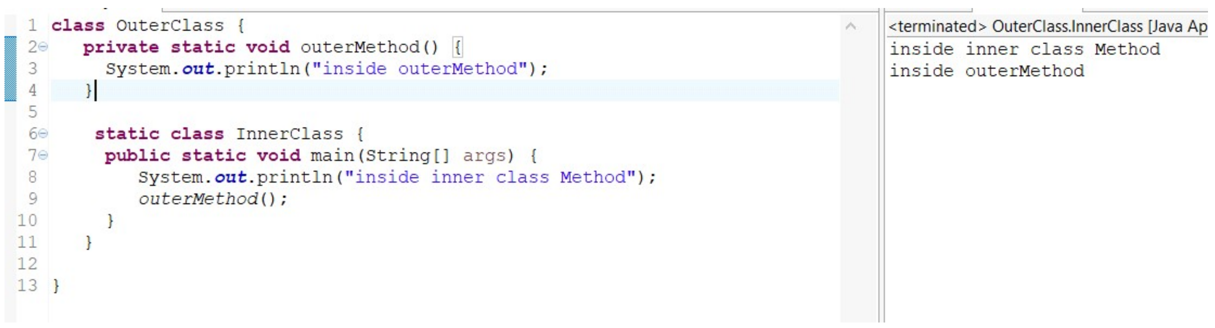
b. The implementer of specified Interface: The anonymous class can extend a class or implement an interface at a time. Here, we have implemented interface in Anonymous class.



```java
interface Anonym {
  void print();
}
class Test {
  static Anonym an = new Anonym() {
    public void print() {
      System.out.println("I am an implementation of interface Anonym");
    }
  };
  public static void main(String[] args) {
    an.print();
  }
}
```

```
<terminated> Test (4) [Java Application] C:\Program Files\Java\jre
I am an implementation of interface Anonym
```

4. Static nested classes: They are more like a static member of the Outer Class.



```java
class OuterClass {
   private static void outerMethod() {
     System.out.println("inside outerMethod");
   }

    static class InnerClass {
     public static void main(String[] args) {
        System.out.println("inside inner class Method");
        outerMethod();
     }
   }

}
```

```
<terminated> OuterClass.InnerClass [Java Ap
inside inner class Method
inside outerMethod
```

Relationship in JAVA

`There are three types of Relationships in JAVA:

1. Dependence (“Uses –A”): This relationship occurs when an object is created inside a method of another class. It is better to make sure that there’s less coupling or a weak dependence of

classes as it can become difficult to manage the program.

Example: Bank uses an Employee.

```
class A{


}
class B{
        void method(){
                A obj = new A();
        }
}
```

2. Association ("Has-A"): This relationship occurs when an object of a class is created as a data member of inside another class. It is easier to understand.

   Example: A vehicle has a motor.

```
class A{


}
class B{
                A obj = new A();
}
```

3. Inheritance ("Is-A"): This relationship occurs between two classes in which one class extends another class.

   Example: A Bus is a vehicle.

```
class A{
}
class B extends A{
}
```