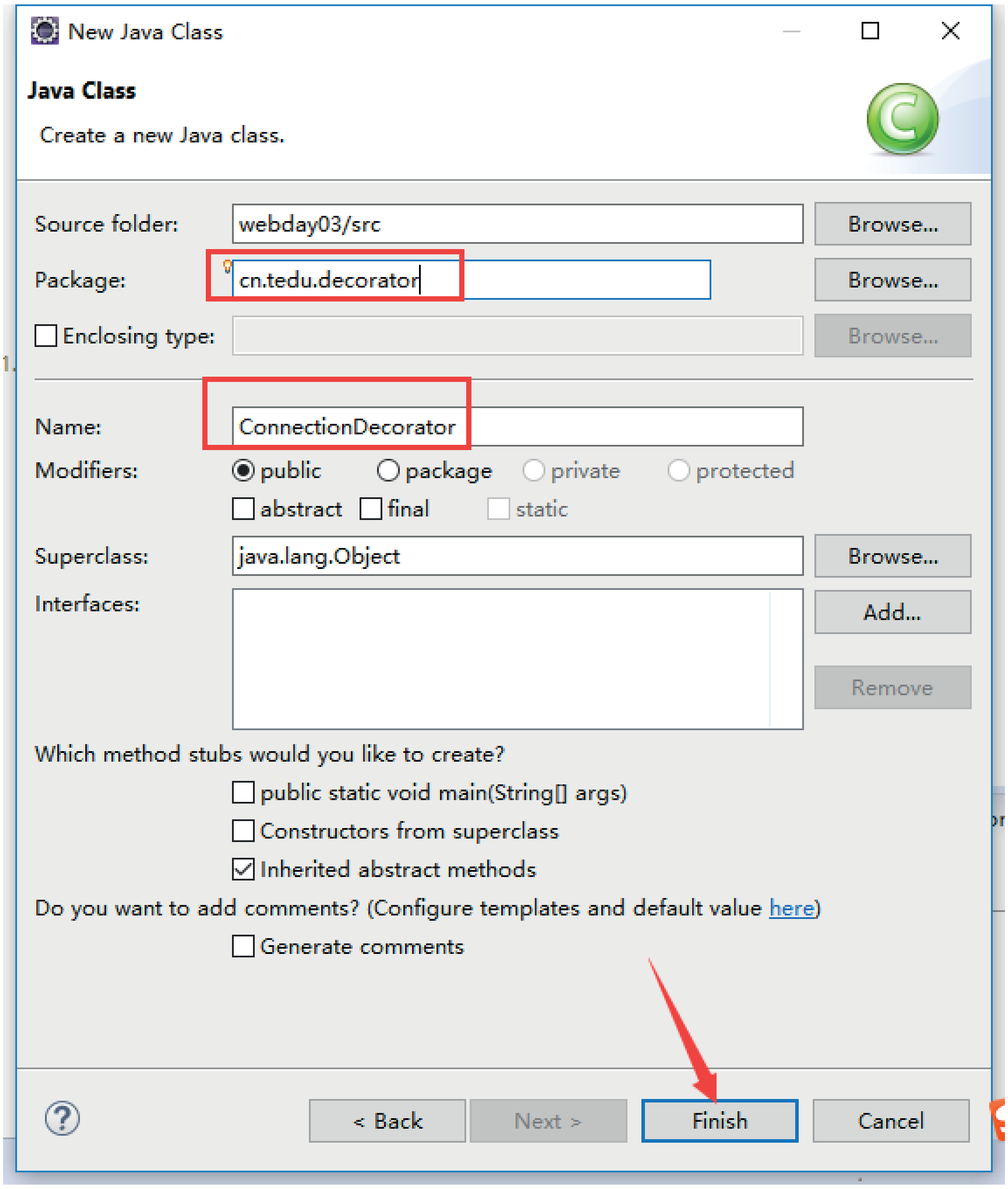
1.2 开发步骤：

1.2.1 写一个包装类 ConnectionDecorator，实现与被包装类 Connection 相同的父类或者接口


New Java Class

Java Class

Create a new Java class.

Source folder: webday03/src

Package: cn.tedu.decorator

☐ Enclosing type:

Name: ConnectionDecorator

Modifiers: ● public ○ package ○ private ○ protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

1.2.2 将被包装类 Connection 传入构造方法中，并且保存在类中

1.2.3 改造你想要改造的方法

```
package cn.tedu.decorator;

import java.sql.Array;
import java.sql.Blob;
import java.sql.CallableStatement;
import java.sql.Clob;
import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.NClob;
import java.sql.PreparedStatement;
import java.sql.SQLClientInfoException;
import java.sql.SQLException;
import java.sql.SQLWarning;
import java.sql.SQLXML;
import java.sql.Savepoint;
import java.sql.Statement;
import java.sql.Struct;
import java.util.Map;
import java.util.Properties;
import java.util.concurrent.Executor;

import cn.tedu.pool.MyPool;

/**
 * 这个类用来包装 Connection 对象的 close 方法
1，实现与被包装类 Connection 相同的父类或者接口
1.2.2  将被包装类 Connection 传入构造方法中，并且保存在类中
1.2.3  改造你想要改造的方法
 */

//1，实现与被包装类 Connection 相同的父类或者接口
public class ConnectionDecorator
```

```
              implements Connection{

    //2,将被包装类 Connection 传入构造方法中,
    //并且保存在类中
    private Connection conn;
    private MyPool pool;
    public ConnectionDecorator(
          Connection conn,MyPool pool){
       this.conn=conn;
       this.pool=pool;
    }
    //3,改造你想要改造的方法
    //改造 close 方法，把连接还回池里
    @Override
    public void close() throws SQLException {
       pool.returnConn(conn);
    }

    //对于不想要改造的方法,
    //保持原有的调用方式
    //删掉不改造的，选中 conn,右键,
    //source,generate delegate methods,OK

    public <T> T unwrap(Class<T> iface) throws SQLException {
       return conn.unwrap(iface);
    }
    public   boolean   isWrapperFor(Class<?>   iface)   throws
SQLException {
       return conn.isWrapperFor(iface);
    }
    public Statement createStatement() throws SQLException {
       return conn.createStatement();
    }
    public PreparedStatement prepareStatement(String sql) throws
SQLException {
       return conn.prepareStatement(sql);
    }
    public  CallableStatement  prepareCall(String  sql)  throws
SQLException {
       return conn.prepareCall(sql);
    }
    public String nativeSQL(String sql) throws SQLException {
       return conn.nativeSQL(sql);
    }
    public   void   setAutoCommit(boolean   autoCommit)   throws
SQLException {
        conn.setAutoCommit(autoCommit);
    }
    public boolean getAutoCommit() throws SQLException {
        return conn.getAutoCommit();
    }
    public void commit() throws SQLException {
        conn.commit();
    }
    public void rollback() throws SQLException {
        conn.rollback();
    }
    public boolean isClosed() throws SQLException {
        return conn.isClosed();
    }
    public DatabaseMetaData getMetaData() throws SQLException {
        return conn.getMetaData();
    }
    public void setReadOnly(boolean readOnly) throws SQLException
{
        conn.setReadOnly(readOnly);
    }
    public boolean isReadOnly() throws SQLException {
        return conn.isReadOnly();
    }
    public void setCatalog(String catalog) throws SQLException {
        conn.setCatalog(catalog);
    }
    public String getCatalog() throws SQLException {
        return conn.getCatalog();
    }
    public  void  setTransactionIsolation(int  level)  throws
SQLException {
        conn.setTransactionIsolation(level);
    }
    public int getTransactionIsolation() throws SQLException {
        return conn.getTransactionIsolation();
    }
    public SQLWarning getWarnings() throws SQLException {
        return conn.getWarnings();
    }
    public void clearWarnings() throws SQLException {
        conn.clearWarnings();
    }
    public  Statement  createStatement(int  resultSetType,  int
resultSetConcurrency) throws SQLException {
        return               conn.createStatement(resultSetType,
resultSetConcurrency);
    }
    public  PreparedStatement  prepareStatement(String  sql,  int
resultSetType, int resultSetConcurrency)
            throws SQLException {
        return     conn.prepareStatement(sql,     resultSetType,
resultSetConcurrency);
    }
    public   CallableStatement   prepareCall(String   sql,   int
resultSetType, int resultSetConcurrency) throws SQLException {
        return       conn.prepareCall(sql,       resultSetType,
resultSetConcurrency);
    }
    public Map<String, Class<?>> getTypeMap() throws SQLException
{
        return conn.getTypeMap();
    }
    public  void  setTypeMap(Map<String,  Class<?>>  map)  throws
SQLException {
        conn.setTypeMap(map);
    }
    public   void   setHoldability(int   holdability)   throws
SQLException {
        conn.setHoldability(holdability);
    }
    public int getHoldability() throws SQLException {
        return conn.getHoldability();
    }
    public Savepoint setSavepoint() throws SQLException {
        return conn.setSavepoint();
    }
    public   Savepoint   setSavepoint(String   name)   throws
SQLException {
        return conn.setSavepoint(name);
    }
    public void rollback(Savepoint savepoint) throws SQLException
{
        conn.rollback(savepoint);
    }
    public  void  releaseSavepoint(Savepoint  savepoint)  throws
SQLException {
        conn.releaseSavepoint(savepoint);
    }
    public  Statement  createStatement(int  resultSetType,  int
resultSetConcurrency, int resultSetHoldability)
            throws SQLException {
        return             conn.createStatement(resultSetType,
resultSetConcurrency, resultSetHoldability);
    }
    public  PreparedStatement  prepareStatement(String  sql,  int
resultSetType, int resultSetConcurrency,
            int resultSetHoldability) throws SQLException {
```

```
        return     conn.prepareStatement(sql,     resultSetType,
resultSetConcurrency, resultSetHoldability);
    }
    public   CallableStatement   prepareCall(String   sql,   int
resultSetType, int resultSetConcurrency,
            int resultSetHoldability) throws SQLException {
        return       conn.prepareCall(sql,       resultSetType,
resultSetConcurrency, resultSetHoldability);
    }
    public  PreparedStatement  prepareStatement(String  sql,  int
autoGeneratedKeys) throws SQLException {
        return conn.prepareStatement(sql, autoGeneratedKeys);
    }
    public PreparedStatement prepareStatement(String sql, int[]
columnIndexes) throws SQLException {
        return conn.prepareStatement(sql, columnIndexes);
    }
    public   PreparedStatement   prepareStatement(String   sql,
String[] columnNames) throws SQLException {
        return conn.prepareStatement(sql, columnNames);
    }
    public Clob createClob() throws SQLException {
        return conn.createClob();
    }
    public Blob createBlob() throws SQLException {
        return conn.createBlob();
    }
    public NClob createNClob() throws SQLException {
        return conn.createNClob();
    }
    public SQLXML createSQLXML() throws SQLException {
        return conn.createSQLXML();
    }
    public boolean isValid(int timeout) throws SQLException {
        return conn.isValid(timeout);
    }
    public void setClientInfo(String name, String value) throws
SQLClientInfoException {
        conn.setClientInfo(name, value);
    }
    public  void  setClientInfo(Properties  properties)  throws
SQLClientInfoException {
        conn.setClientInfo(properties);
    }
    public String getClientInfo(String name) throws SQLException
{
        return conn.getClientInfo(name);
    }
    public Properties getClientInfo() throws SQLException {
        return conn.getClientInfo();
    }
    public   Array   createArrayOf(String   typeName,   Object[]
elements) throws SQLException {
        return conn.createArrayOf(typeName, elements);
    }
    public   Struct   createStruct(String   typeName,   Object[]
attributes) throws SQLException {
        return conn.createStruct(typeName, attributes);
    }
    public void setSchema(String schema) throws SQLException {
        conn.setSchema(schema);
    }
    public String getSchema() throws SQLException {
        return conn.getSchema();
    }
    public void abort(Executor executor) throws SQLException {
        conn.abort(executor);
    }
    public   void   setNetworkTimeout(Executor   executor,   int
milliseconds) throws SQLException {
        conn.setNetworkTimeout(executor, milliseconds);
    }
    public int getNetworkTimeout() throws SQLException {
        return conn.getNetworkTimeout();
    }


}
```

1.2.4 改造 MyPool 类

```
//3,提供 getConnection 方法，对外提供获取连接对象
    @Override
    public Connection getConnection() throws SQLException {
//      list.get(0);//只是拿出来看看
        //拿出来看看，从池子里删除掉这个连接
        Connection conn = list.remove(0);
        System.out.println("连接少了一个...");

        //创建包装类的对象
```