Internet advertisement Auction

Objected Oriented Programming

Mykhailo Sichkaruk

Table of Contents

Main criteria:

Classes:

```
/**
 * Model of Auction that can be used to implement own Auction type
 */
public class AbstractModel {
```

This class is root to the project`s backend. This class provides nested class that represents user`s data in object format. Here is Encapsulation comes in, UserData variables is inaccessible to set new values, but can be returned with Getters.

```
 public static class UserData {
        private int id;
        private String login;
        private String email;
        private int balance;
        private String mode;
        private String license;

        public UserData(int userId) throws SQLException {
            ResultSet res = SQL.SELECT_UserData(userId);
            res.next();
            this.login = res.getString(Const.SQL.USERDATA_LOGIN);
            this.balance = res.getInt(Const.SQL.USERDATA_BALANCE);
            this.email = res.getString(Const.SQL.USERDATA_EMAIL);
            this.mode = res.getString(Const.SQL.USERDATA_MODE);
            this.license = res.getString(Const.SQL.USERDATA_LICENSE);
            this.id = userId;
        }

        /**
         * @return the id
         */
        public int getId() {
            return id;
        }

        /**
         * @param id the id to set
         */
        private void setId(int id) {
            this.id = id;
        }
```

```java
    /**
     * @return the login
     */
    public String getLogin() {
        return login;
    }

    /**
     * @param login the login to set
     */
    private void setLogin(String login) {
        this.login = login;
    }

    /**
     * @return the email
     */
    public String getEmail() {
        return email;
    }

    /**
     * @param email the email to set
     */
    private void setEmail(String email) {
        this.email = email;
    }

    /**
     * @return the balance
     */
    public int getBalance() {
        return balance;
    }

    /**
     * @param balance the balance to set
     */
    private void setBalance(int balance) {
        this.balance = balance;
    }

    /**
     * @return the mode
```

```
         */
        public String getMode() {
            return mode;
        }

        /**
         * @param mode the mode to set
         */
        private void setMode(String mode) {
            this.mode = mode;
        }

        /**
         * @return the license
         */
        public String getLicense() {
            return license;
        }

        /**
         * @param license the license to set
         */
        private void setLicense(String license) {
            this.license = license;
        }
    }
```

Main idea of this class is that it can be starting point in creating Model of auction. And this is exactly what it is doing in my project.

Auction.java

The

```
public class Auction extends AbstractModel{
```

is a class that implements real model of auction process.

Functionality:

Can end Auction and calculate the winner:

```
    public static void endAuction(int lotId) throws SQLException {
```

Verify license of the user:

```
 public static boolean verifyLicense(File file) throws SQLException, IOException
```

Handle your try of adding new bid:

```
    public static boolean tryAddBid(String bidStr, int lotId) throws
SQLException, BidException {
```

Main Criteria:

Inheritance:

1)

The

```
public class Auction extends AbstractModel{
```

is a class that implements real model of auction process.

Idea: We can extend `AbstractModel` to create our own implementations of Auction.

2)

```
public class User implements Handler {
```

and

```
public class Seller extends User {
```

```
public class Buyer extends User {
```

```
public class Auctioner extends User {
```

Idea: User can : EndAuction, AddLot, AddBid

But,

`Seller` can only AddLot

`Buyer` can only AddBid

`Auctioner` can only EndAuction

So they extend only parts, they need. Other features is unavailable.

Polymorphism:

User, Auctioner, Seller, Buyer have different implementations of initialize() function.

Idea: Every class provides specified features by disabling unnecessary features.

User: Set focus on AddLotInput | set listener to EndAuctionButton | Prints Lots | formats input of AddBidInput (only numbers)

```
    /**
     * Initialize.
     *
     * @throws SQLException the sql exception
     */
    public void initialize() throws SQLException {
        Vbox_lots.setBackground(new Background(new
BackgroundFill(Color.DARKGREEN, CornerRadii.EMPTY, Insets.EMPTY)));
        scroll_lots.setStyle("-fx-background: DARKSLATEGREY; -fx-border-color:
#90EE90;");
        printLots();
        Platform.runLater(addLotInput::requestFocus);
        updateUserData();
        class EndAuctionHandler implements EventHandler<ActionEvent> {
            @Override
            public void handle(ActionEvent event) {
                if (lotCheckedID != -1) {
                    try {
                        Auction.endAuction(lotCheckedID);
                    } catch (SQLException e) {
                        e.printStackTrace();
                    }
                    if (!Auction.isEndAuctionFirstClick()) {
                        try {
                            printLots();
                            updateUserData();
                        } catch (SQLException e) {
                            e.printStackTrace();
                        }
                    }

                }
            }
        }
        endAuctionButton.setOnAction(new EndAuctionHandler());
```

```
        addBidInput.textProperty().addListener(new ChangeListener<String>() {
            @Override
            public void changed(ObservableValue<? extends String> observable,
String oldValue,
                    String newValue) {
                if (!newValue.matches("\\d*")) {
                    addBidInput.setText(newValue.replaceAll("[^\\d]", ""));
                }
            }
        });
}
```

Auctioner : Disables addBidInput and addLotInput | shows “BuyPro” banner | And shows text about unavailable features.

```
    /**
     * Disables feachures of [Adding Lot] and [Adding Bid]
     * Adds "BuyPro" Banner
     */
    @Override
    public void initialize() throws SQLException {
        super.initialize();

        addLotInput.setPromptText("This is PRO feachure");
        addBidInput.setPromptText("You can buy PRO version to use all feachures
in one account");
        addLotInput.setDisable(true);
        addBidInput.setDisable(true);
        setProBanner();
    }
```

Seller: Disables addBidInput and EndAuctionButton | shows “BuyPro” banner | And shows text about unavailable features.

```
    /**
     * Disables feachures of [Adding Bid] and [Ending Auction]
     * Adds "BuyPro" Banner
     */
    public void initialize() throws SQLException {
        super.initialize();

        addBidInput.setPromptText("You can buy PRO version to use all feachures
in one account");
        addBidInput.setDisable(true);
        endAuctionButton.setDisable(true);
```

```
        proBannerGrid.setVisible(true);
        setProBanner();
    }
```

Buyer : Disables addLotInput and EndAuctionButton | shows “BuyPro” banner | And shows text about unavailable features.

```
    /**
     * Disables feachures of [Adding Lot] and [Ending Auction]
     * Adds "BuyPro" Banner
     */
    @Override
    public void initialize() throws SQLException {
        super.initialize();

        addLotInput.setPromptText("Try PRO version to use this");
        addLotInput.setDisable(true);
        endAuctionButton.setDisable(true);

        Platform.runLater(addBidInput::requestFocus);
        setProBanner();
    }
```

Polymorphism 2:

Auction extends AbstractModel and overrides updateUser() method, because in this implementation of Auction it isn't always necessary to put newUserId to update Userdata.

AbstractModel:

```
    /**
     * Update user.
     *
     * @param newUserId the new user id
     * @throws SQLException the sql exception
     */
    public static void updateUser(int newUserId) throws SQLException {
        setUSER(new UserData(newUserId));
    }
```

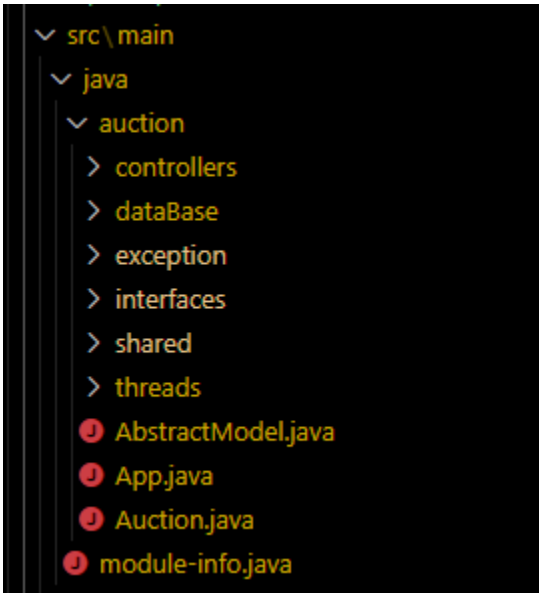
Auction:

```
    public static void updateUser() throws SQLException {

        setUSER(new UserData(userId));
    }
```

Packages:

Code divided into packages to rich better modularity and less coupling.



Encapsulation:

Userdata class have private setters except setId(int id). So nobody can changes the inner data of this class.

```
public static class UserData {
        private int id;
        private String login;
        private String email;
        private int balance;
        private String mode;
        private String license;

        /**
         * Instantiates a new User data.
         *
         * @param userId the user id
         * @throws SQLException the sql exception
         */
        public UserData(int userId) throws SQLException {
            ResultSet res = SQL.SELECT_UserData(userId);
            res.next();
            this.login = res.getString(Const.SQL.USERDATA_LOGIN);
            this.balance = res.getInt(Const.SQL.USERDATA_BALANCE);
            this.email = res.getString(Const.SQL.USERDATA_EMAIL);
            this.mode = res.getString(Const.SQL.USERDATA_MODE);
            this.license = res.getString(Const.SQL.USERDATA_LICENSE);
            this.id = userId;
        }

        /**
         * Gets id.
         *
         * @return the id
         */
        public int getId() {
            return id;
        }

        /**
         * @param id the id to set
         */
        private void setId(int id) {
            this.id = id;
        }
```

```java
    /**
     * Gets login.
     *
     * @return the login
     */
    public String getLogin() {
        return login;
    }

    /**
     * @param login the login to set
     */
    private void setLogin(String login) {
        this.login = login;
    }

    /**
     * Gets email.
     *
     * @return the email
     */
    public String getEmail() {
        return email;
    }

    /**
     * @param email the email to set
     */
    private void setEmail(String email) {
        this.email = email;
    }

    /**
     * Gets balance.
     *
     * @return the balance
     */
    public int getBalance() {
        return balance;
    }

    /**
     * @param balance the balance to set
     */
    private void setBalance(int balance) {
```

```
            this.balance = balance;
        }

        /**
         * Gets mode.
         *
         * @return the mode
         */
        public String getMode() {
            return mode;
        }

        /**
         * @param mode the mode to set
         */
        private void setMode(String mode) {
            this.mode = mode;
        }

        /**
         * Gets license.
         *
         * @return the license
         */
        public String getLicense() {
            return license;
        }

        /**
         * @param license the license to set
         */
        private void setLicense(String license) {
            this.license = license;
        }
    }
```

Aggregation:

In Auction class I use instance of UserData class to save data of current user.

Auction #42

```
    private static UserData currentUser;
```

Auction #173

```
    /**
     * Update user.
     *
     * @throws SQLException the sql exception
     */
    public static void updateUser() throws SQLException {
        setUSER(new UserData(userId));
    }
```

Also I use Aggregation when I create own threads:

Auction #237

```
    /**
     * Creates LicenseKey
     * Updates DataBase
     * Switches window to PRO mode
     *
     * @throws NoSuchAlgorithmException the no such algorithm exception
     * @throws SQLException             the sql exception
     * @throws IOException              the io exception
     */
    public static void setLicenseKey() throws NoSuchAlgorithmException,
SQLException, IOException {
        String licenseKey = generateLicenseKey();
        Thread sendLicenseThread = new SendLicenseEmail(licenseKey,
currentUser.getEmail(), currentUser.getLogin());
        sendLicenseThread.start();
        //Update license record in SQL
        Thread updateLicense = new UpdateLicense(licenseKey,
currentUser.getId());
        updateLicense.start();
```

Further criteria:

Serialization:

I wrote License Key of Pro version into JSON file and sent via Email, and read it from filesystem to verify Pro version owner

Deserialization:

In Auction #315

```
    private static String getLicenseFromJSON(File file) throws
FileNotFoundException{
        String path = currentUser.getLogin() + ".json";
        if(file != null)
            path = file.getPath();

        JSONParser jsonParser = new JSONParser();
        String licenseKey = "";
        try {
            JSONObject licenseKeyJSON = (JSONObject) jsonParser.parse(new
FileReader(path));
             licenseKey = (String) licenseKeyJSON.get("key");
        } catch (IOException|ParseException e) {
            System.out.println("Cannot find license near app, please try add
license manually");
        }

        return licenseKey;
    }
```

Serialization:

In auction.threads. SendLicenseEmail #105

```
    private void createJSON(){
        JSONObject licenseJSON = new JSONObject();
        licenseJSON.put("key", licenseKey);
        licenseJSON.put("login", login);
        try {
            FileWriter file = new FileWriter(this.login + ".json");
            file.write(licenseJSON.toJSONString());
            System.out.println("file writed");
            file.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
```

Default method implementation in interfaces:

In auction.controllers.Handler #21

```
    /**
     * Sign out handle.
     *
     * @param ke the ke
     * @throws IOException the io exception
     */
    public default void signOutHandle(KeyEvent ke) throws IOException {
        if (ke.getCode().equals(KeyCode.ESCAPE)) {
            App.setRoot(Const.FXML.LOGIN_SCENE);
        }
    }
```

It will invokes if user will press ESC key in GUI

Lambda expression:

Auction #268

```
    /**
     * Here we specify how our message will be created
     * message is a text, that will be hashed and set as License
     * so it is importand to set some algorithm that combines some parts ow
information and returns message
     *
     * right now here I have used simple concatenation, but it could also be
like:
     * arg1 + arg1.lenght()/2 + arg2.substring(3) + arg3.charAt(10);
     */
    private static void setEncryprionMessage(){
        message = (arg1, arg2, arg3)-> {
            return arg1 + arg2 + arg3;
        };
    }
```

Auction #48

```
    static License message;
```

auction.interfaces.License

```
/**
 * The interface License.
 */
```

```
public interface License {
    /**
     * Encryprion message string.
     *
     * @param arg1 the arg 1
     * @param arf2 the arf 2
     * @param arg3 the arg 3
     * @return the string
     */
    public String encryprionMessage(String arg1, String arf2, String arg3);
}
```

Nested Class:

AbstractModel #18

```
/**
     * Nested Class that will be used in Model to save User`s data in a Obbject
*/
public static class UserData {
        private int id;
        private String login;
        private String email;
        private int balance;
        private String mode;
        private String license;
```

RTTI:

We need to know what class is managing now to put actual “BUY PRO” banner.

auction.controllers.User #165

```
    /**
     * Shows custom banner depends on user`s mode
     */
    protected void setProBanner() {
        // RTTI implmentation
        if (this.getClass() == (new Buyer()).getClass())
            currentVersionLable.setText("Buyer");
        if (this.getClass() == (new Auctioner()).getClass())
            currentVersionLable.setText("Auctioner");
        if (this.getClass() == (new Seller()).getClass())
            currentVersionLable.setText("Sellers");

        proBannerGrid.setVisible(true);
```

```
    }
```

Multithreading:

Parallel sending Email and updating LicenseKey in SQL

```
    /**
     * Creates LicenseKey
     * Updates DataBase
     * Switches window to PRO mode
     *
     * @throws NoSuchAlgorithmException the no such algorithm exception
     * @throws SQLException             the sql exception
     * @throws IOException              the io exception
     */
    public static void setLicenseKey() throws NoSuchAlgorithmException,
SQLException, IOException {
        String licenseKey = generateLicenseKey();
        Thread sendLicenseThread = new SendLicenseEmail(licenseKey,
currentUser.getEmail(), currentUser.getLogin());
        sendLicenseThread.start();
        //Update license record in SQL
        Thread updateLicense = new UpdateLicense(licenseKey,
currentUser.getId());
        updateLicense.start();
        App.changeScene(Const.FXML.AUCTION_SCENE, new User());
    }
```

GUI separated from Logic:

Controllers responsible for control of FXML = GUI

Auction is backend, where everything except GUI is running

Manually created Handlers:

Create own handler than invokes Auction.endAuction() method and binds it to endAuctionButton

auction.controllers.User #127

```
class EndAuctionHandler implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent event) {
        if (lotCheckedID != -1) {
            try {
                Auction.endAuction(lotCheckedID);
            } catch (SQLException e) {
                e.printStackTrace();
            }
            if (!Auction.isEndAuctionFirstClick()) {
                try {
                    printLots();
                    updateUserData();
                } catch (SQLException e) {
                    e.printStackTrace();
                }
            }

        }
    }
```

```
        }
        endAuctionButton.setOnAction(new EndAuctionHandler());
```

Another version of adding listener that specifies input in AddBidInput (only number)

```
        addBidInput.textProperty().addListener(new ChangeListener<String>() {
            @Override
            public void changed(ObservableValue<? extends String> observable,
String oldValue,
                    String newValue) {
                if (!newValue.matches("\\d*")) {
                    addBidInput.setText(newValue.replaceAll("[^\\d]", ""));
                }
            }
        });
```

Own exceptions:

I throw new own exception if entered bid is [EMPTY] or [0]

Declaration

```
package auction.exception;

/**
 * Custom exception
 */
public class BidException extends Exception
{
    /**
     * Exception about bid input
     *
     * @param str the str
     */
    public BidException (String str)
    {
        super(str);
    }
}
```

Usage:

Auction #60

```
    public static boolean tryAddBid(String bidStr, int lotId) throws
SQLException, BidException {
        if (bidStr.equals("")) {
            throw new BidException("You cannot add [EMPTY] bid");
        } else if (bidStr.equals("0")) {
            throw new BidException("You cannot add [0] bid");
        } else {
```