BearPool Ride-Sharing Iteration 3

Created by:
Joshua Huertas
Andrew Ammentorp
Leighton Glim
Joseph Perez
Joseph Yu

Architecture Design

Business

- ❖ CreatePostValidate
- ❖ IService
- ❖ Login
- ❖ PostService
- ❖ ReportService
- ❖ SurveyService
- ❖ UserService
- ❖ ValidateAccountInfo
- ❖ ValidateSurvey

Data

- databaseControllers
    - ❖ Database
    - ❖ PostDatabase
    - ❖ ReportDatabase
    - ❖ SurveyDatabase
    - ❖ UserDatabase

- post
    - ❖ AbstractPost
    - ❖ Driver
    - ❖ Rider

- survey
    - ❖ Survey

- users
    - ❖ Admin
    - ❖ Report
    - ❖ User

Enums

- ❖ Failures
- ❖ Ratings

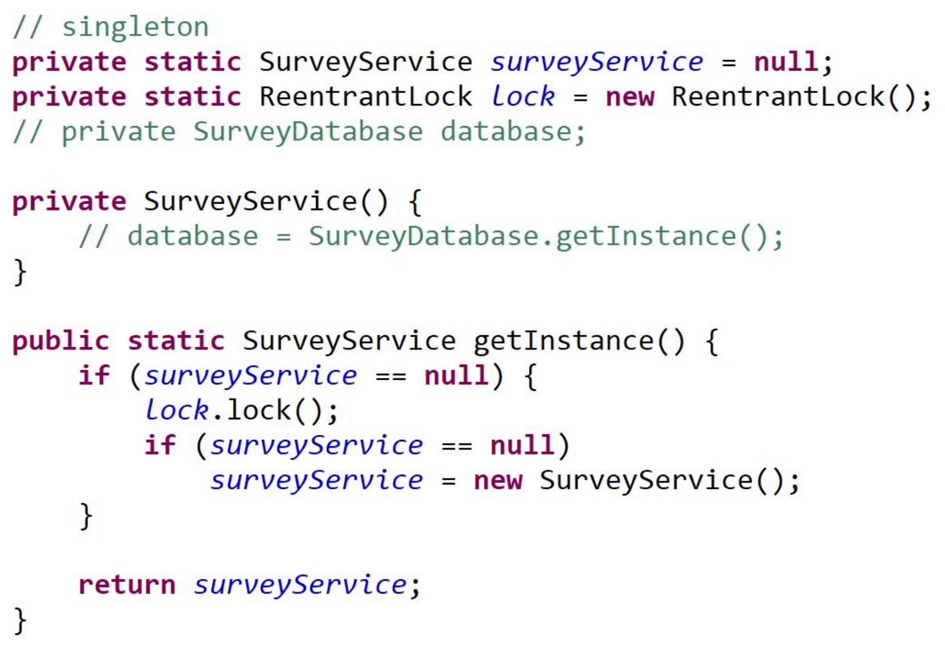
Presentation

- application
    - AccountCreateDialog
    - AdminReport
    - Application
    - CreateDriverTable
    - CreateMyRidesTable
    - CreatePost
    - CreateRiderTable
    - EditProfile
    - LoginDialog
    - OpenPage
    - RowFilterUI
    - SelectPostType
    - SurveyGUI
    - ViewPostInfo
    - ViewProfile

Design Patterns

Singleton

- All services (UserService, PostService, SurveyService, ReportSurvey) and the databases(UserDatabase, PostDatabase, SurveyService, ReportDatabase)
- Only one instance of these objects is needed to perform their jobs, so each one was made to be a singleton. When accessed by other classes, the classes return an instance of themselves so all classes using them are using the same instance.



```
// singleton
private static SurveyService surveyService = null;
private static ReentrantLock lock = new ReentrantLock();
// private SurveyDatabase database;

private SurveyService() {
    // database = SurveyDatabase.getInstance();
}

public static SurveyService getInstance() {
    if (surveyService == null) {
        lock.lock();
        if (surveyService == null)
            surveyService = new SurveyService();
    }

    return surveyService;
}
```

Mediator

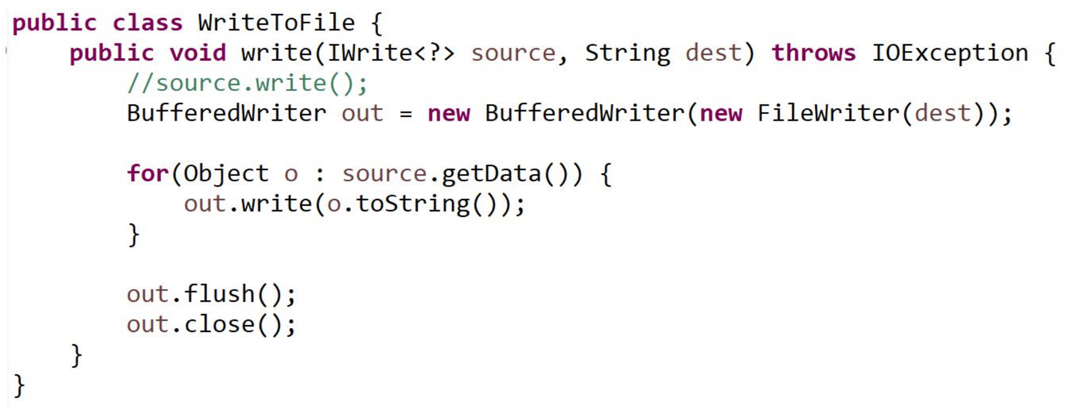
- All Services (UserService, PostService, SurveyService, ReportService)
- Communication with all the presentation level classes and the databases must go through the Services first. In this way, they act as mediators, as the business logic cannot work with the database logic without the services.

Chain of responsibility

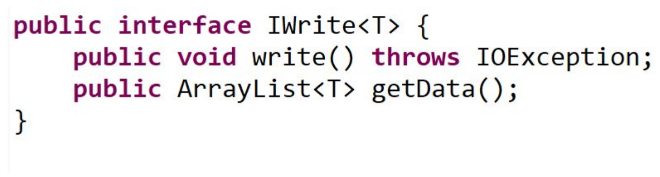
- Many of the UI classes act as command objects, and the services and databases act as processing objects
- A UI class will call a method in a service object. The service object will do the work requested by the command object until it cannot anymore due to lack of information. At this point the command is passed down to the database level, where it is completed.

Servant

- All Databases (AbstractDatabase, UserDatabase, PostDatabase, SurveyService, ReportDatabase), WriteToFile, and IWrite
- While Writing the database information to output files, the database itself shouldn't have to handle the operation itself. The task of writing to a file destination is performed by the WriteToFile class combining with the interface IWrite. All four Database classes can all be written to files by the usage of the IWrite interface.



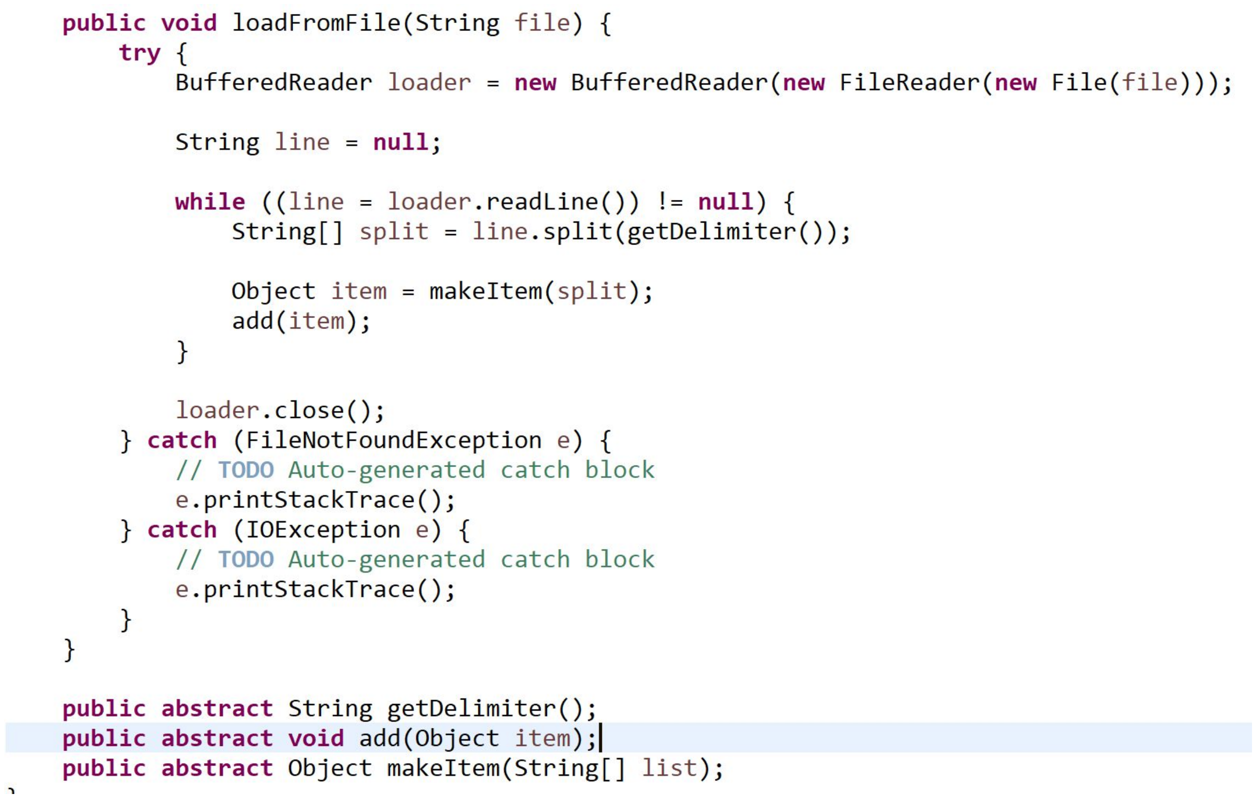
```
public class WriteToFile {
    public void write(IWrite<?> source, String dest) throws IOException {
        //source.write();
        BufferedWriter out = new BufferedWriter(new FileWriter(dest));

        for(Object o : source.getData()) {
            out.write(o.toString());
        }

        out.flush();
        out.close();
    }
}
```



```
public interface IWrite<T> {
    public void write() throws IOException;
    public ArrayList<T> getData();
}
```

Template Method

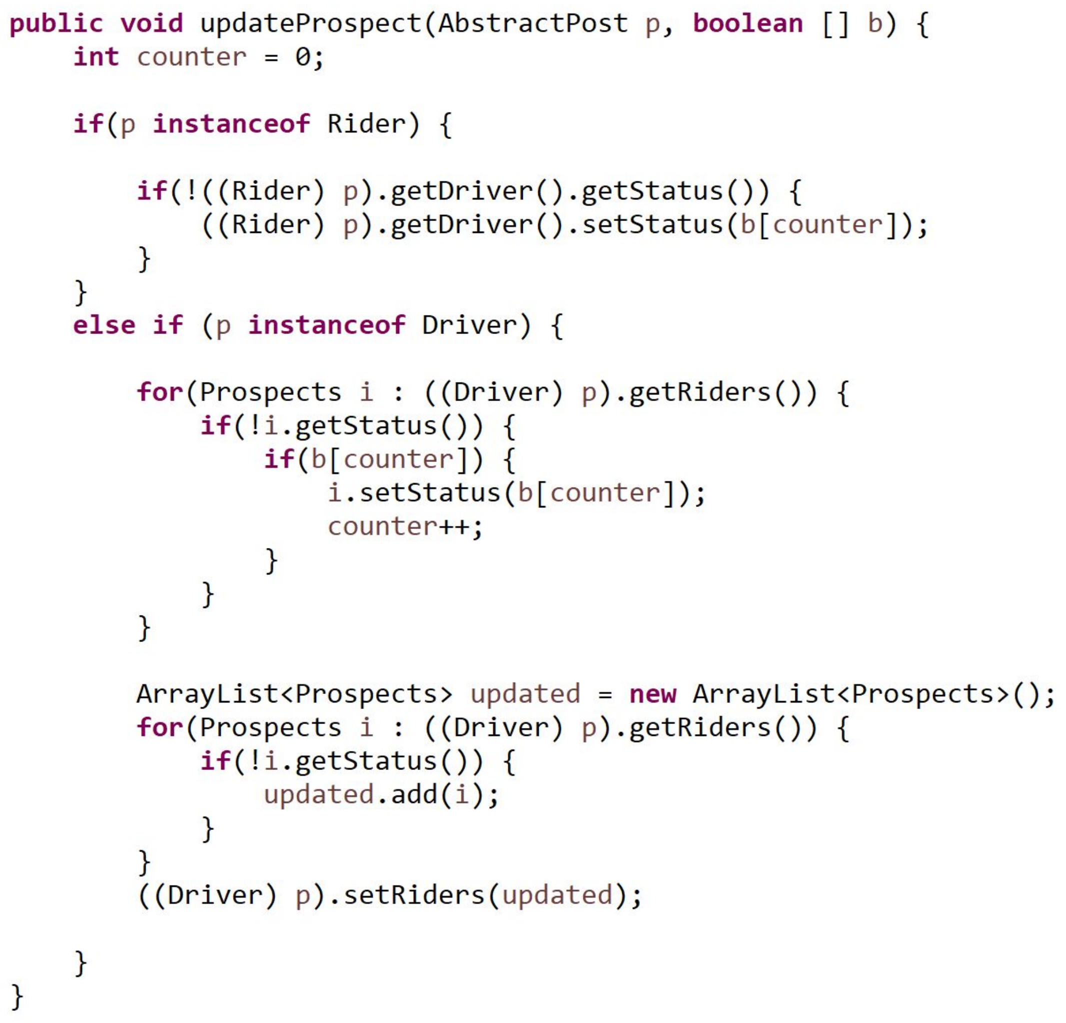
- AbstractDatabase, UserDatabase, PostDatabase, SurveyService, ReportDatabase
- The operation that loads data from a file is similar and the general algorithm can be reduced and generalized as a template method



```java
public void loadFromFile(String file) {
    try {
        BufferedReader loader = new BufferedReader(new FileReader(new File(file)));

        String line = null;

        while ((line = loader.readLine()) != null) {
            String[] split = line.split(getDelimiter());

            Object item = makeItem(split);
            add(item);
        }

        loader.close();
    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

public abstract String getDelimiter();
public abstract void add(Object item);
public abstract Object makeItem(String[] list);
```

Strategy

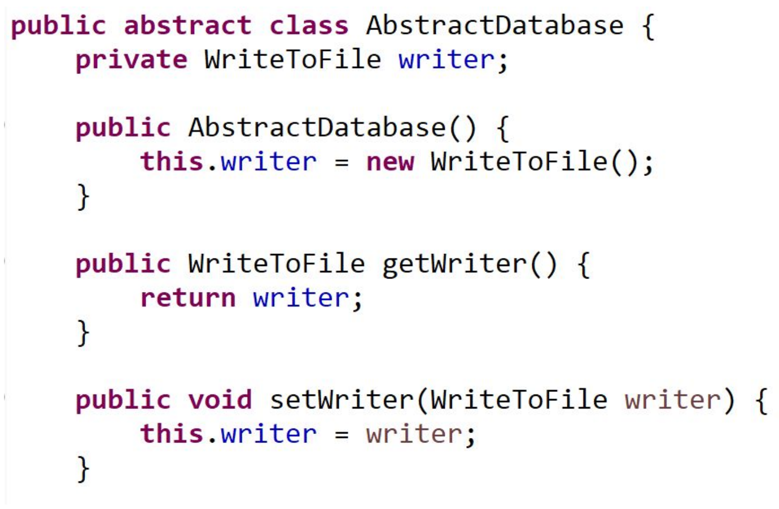
- The updateProspect Method in the postService is an example of Strategy.
- Update Prospect can use either a rider object or a driver object and depend on the object given at run time can run 2 different algorithms to store the required information in the database.



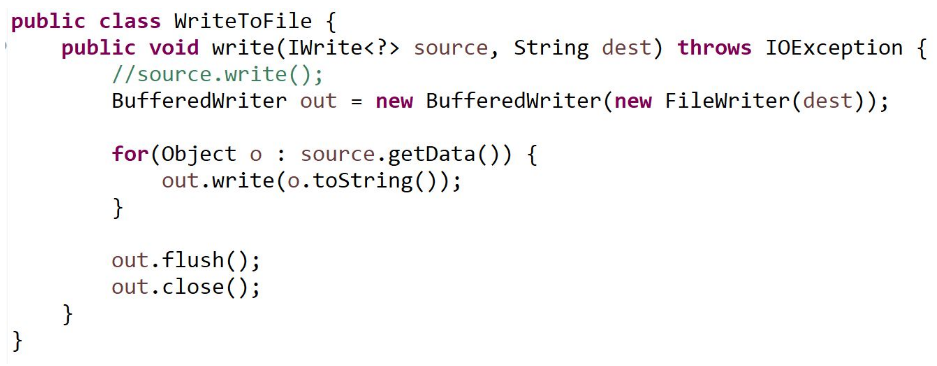
```java
public void updateProspect(AbstractPost p, boolean [] b) {
    int counter = 0;

    if(p instanceof Rider) {

        if(!((Rider) p).getDriver().getStatus()) {
            ((Rider) p).getDriver().setStatus(b[counter]);
        }
    }
    else if (p instanceof Driver) {

        for(Prospects i : ((Driver) p).getRiders()) {
            if(!i.getStatus()) {
                if(b[counter]) {
                    i.setStatus(b[counter]);
                    counter++;
                }
            }
        }

        ArrayList<Prospects> updated = new ArrayList<Prospects>();
        for(Prospects i : ((Driver) p).getRiders()) {
            if(!i.getStatus()) {
                updated.add(i);
            }
        }
        ((Driver) p).setRiders(updated);

    }
}
```

Bridge

- All Databases (UserDatabase, PostDatabase, SurveyService, ReportDatabase), WriteToFile, IWrite, AbstractDatabase
- The WriteToFile member that exists in the database classes separates the responsibility for the databases to record its data. The databases are then able to have different behaviors regarding the representation of their information.



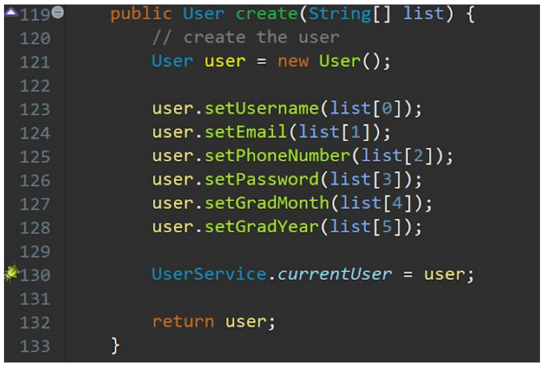
```java
public abstract class AbstractDatabase {
    private WriteToFile writer;

    public AbstractDatabase() {
        this.writer = new WriteToFile();
    }

    public WriteToFile getWriter() {
        return writer;
    }

    public void setWriter(WriteToFile writer) {
        this.writer = writer;
    }
```



```java
public class WriteToFile {
    public void write(IWrite<?> source, String dest) throws IOException {
        //source.write();
        BufferedWriter out = new BufferedWriter(new FileWriter(dest));

        for(Object o : source.getData()) {
            out.write(o.toString());
        }

        out.flush();
        out.close();
    }
}
```

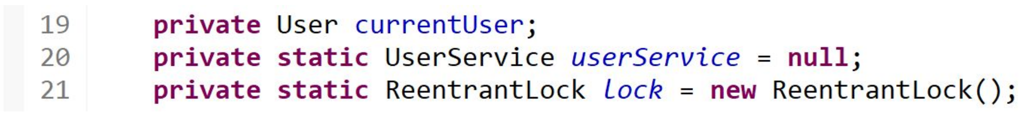
Code Analysis - SpotBugs

- UserService:
  - Line 130: Write to static field business.UserService.currentUser from instance method business.UserService.create(String[]) [Of Concern(15), High confidence]



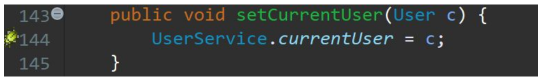
```
119     public User create(String[] list) {
120         // create the user
121         User user = new User();
122 
123         user.setUsername(list[0]);
124         user.setEmail(list[1]);
125         user.setPhoneNumber(list[2]);
126         user.setPassword(list[3]);
127         user.setGradMonth(list[4]);
128         user.setGradYear(list[5]);
129 
130         UserService.currentUser = user;
131 
132         return user;
133     }
```

    - Fixed: changed UserService.currentUser from static to normal. This does not affect functionality because we only have one instance of UserService since it is a singleton.



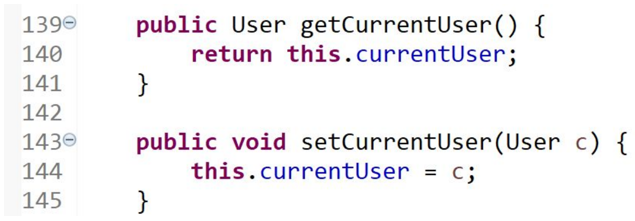
```
19      private User currentUser;
20      private static UserService userService = null;
21      private static ReentrantLock lock = new ReentrantLock();
```

  - Line 144: Write to static field business.UserService.currentUser from instance method business.UserService.setCurrentUser(User) [Of Concern(15), High confidence]



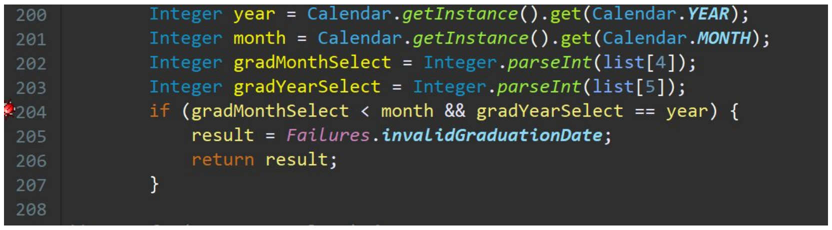
```
143     public void setCurrentUser(User c) {
144         UserService.currentUser = c;
145     }
```

    - Fixed: The action of changing UserService.currentUser from static to normal changed resolved the issue.



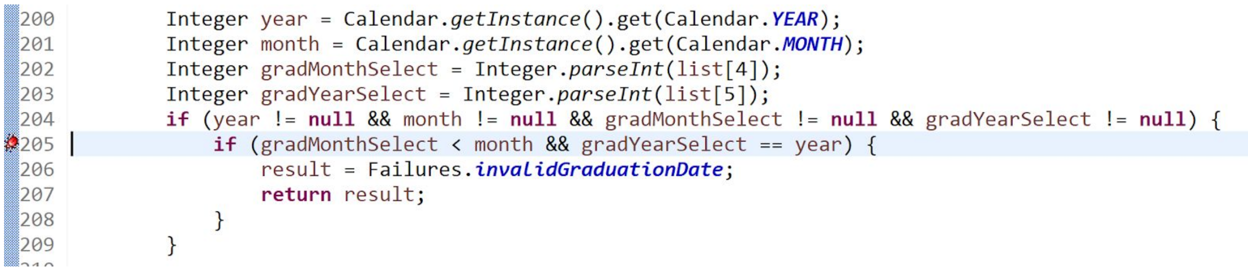
```
139     public User getCurrentUser() {
140         return this.currentUser;
141     }
142 
143     public void setCurrentUser(User c) {
144         this.currentUser = c;
145     }
```

- Line 204: Suspicious comparison of Integer references in business.UserService.update(String[]) [Scariest(1), High confidence]



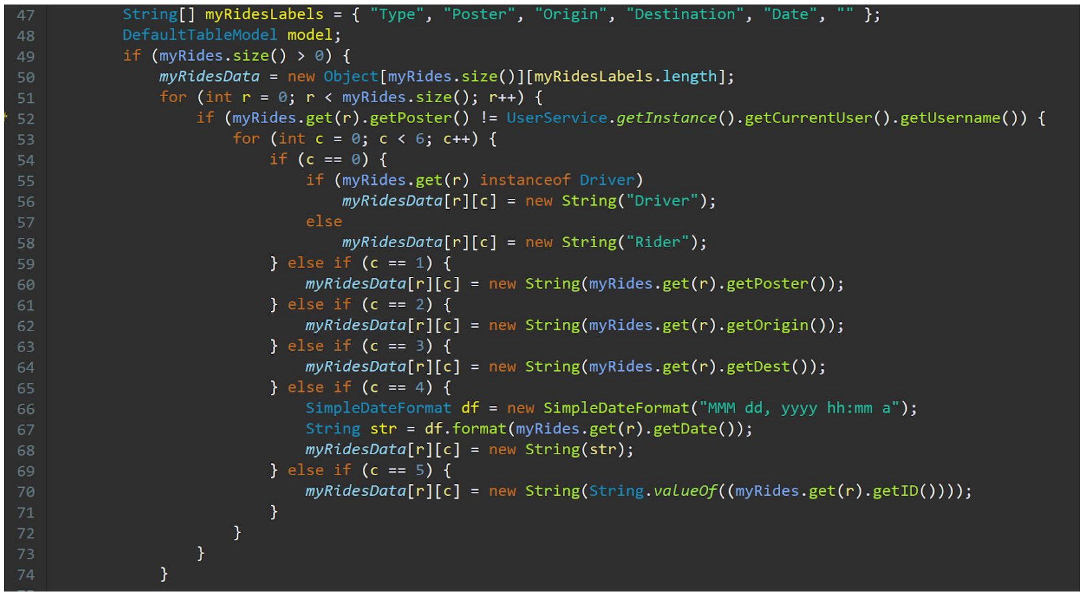
```
200         Integer year = Calendar.getInstance().get(Calendar.YEAR);
201         Integer month = Calendar.getInstance().get(Calendar.MONTH);
202         Integer gradMonthSelect = Integer.parseInt(list[4]);
203         Integer gradYearSelect = Integer.parseInt(list[5]);
204         if (gradMonthSelect < month && gradYearSelect == year) {
205             result = Failures.invalidGraduationDate;
206             return result;
207         }
208
```

  - There was a strange problem with this bug. We understood that the Integers we are comparing could be null, so we added an if clause to ensure that they are not. However, this did not get rid of the bug when spot bugs scanned it despite it being impossible for null values to be compared now.



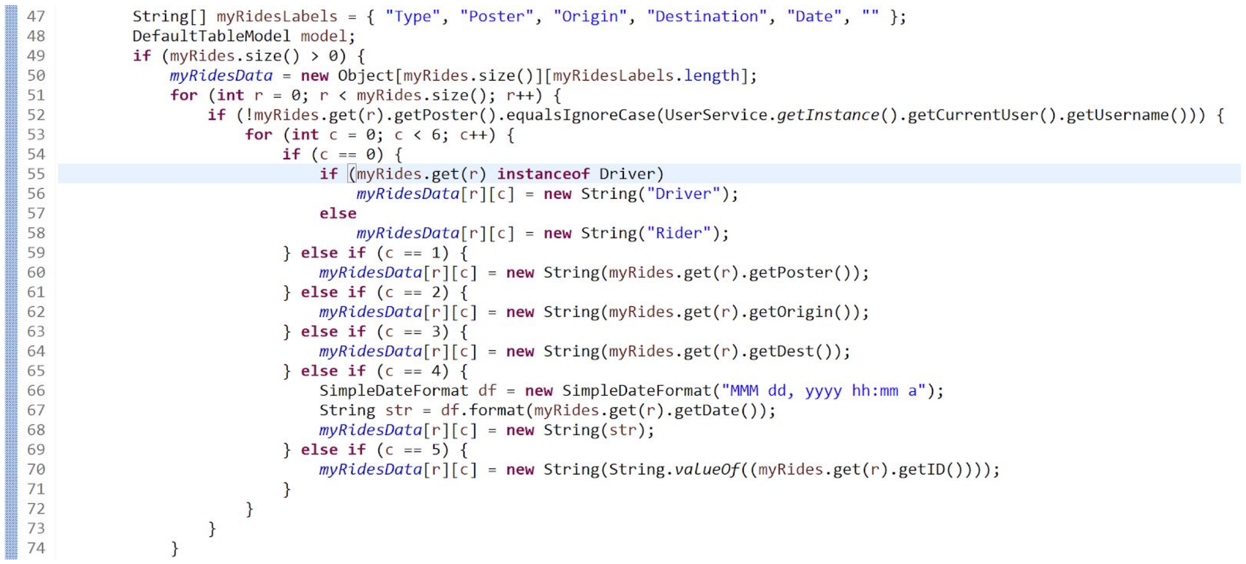
```
200         Integer year = Calendar.getInstance().get(Calendar.YEAR);
201         Integer month = Calendar.getInstance().get(Calendar.MONTH);
202         Integer gradMonthSelect = Integer.parseInt(list[4]);
203         Integer gradYearSelect = Integer.parseInt(list[5]);
204         if (year != null && month != null && gradMonthSelect != null && gradYearSelect != null) {
205             if (gradMonthSelect < month && gradYearSelect == year) {
206                 result = Failures.invalidGraduationDate;
207                 return result;
208             }
209         }
```

- CreateMyRidesTable:
    - Line 52: Comparison of String objects using == or != in presentation.application.CreateMyRidesTable.createTable(ArrayList) [Troubling(11), Normal confidence]



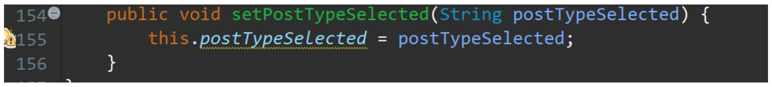
```
47         String[] myRidesLabels = { "Type", "Poster", "Origin", "Destination", "Date", "" };
48         DefaultTableModel model;
49         if (myRides.size() > 0) {
50             myRidesData = new Object[myRides.size()][myRidesLabels.length];
51             for (int r = 0; r < myRides.size(); r++) {
52                 if (myRides.get(r).getPoster() != UserService.getInstance().getCurrentUser().getUsername()) {
53                     for (int c = 0; c < 6; c++) {
54                         if (c == 0) {
55                             if (myRides.get(r) instanceof Driver)
56                                 myRidesData[r][c] = new String("Driver");
57                             else
58                                 myRidesData[r][c] = new String("Rider");
59                         } else if (c == 1) {
60                             myRidesData[r][c] = new String(myRides.get(r).getPoster());
61                         } else if (c == 2) {
62                             myRidesData[r][c] = new String(myRides.get(r).getOrigin());
63                         } else if (c == 3) {
64                             myRidesData[r][c] = new String(myRides.get(r).getDest());
65                         } else if (c == 4) {
66                             SimpleDateFormat df = new SimpleDateFormat("MMM dd, yyyy hh:mm a");
67                             String str = df.format(myRides.get(r).getDate());
68                             myRidesData[r][c] = new String(str);
69                         } else if (c == 5) {
70                             myRidesData[r][c] = new String(String.valueOf((myRides.get(r).getID())));
71                         }
72                     }
73                 }
74             }
```

- Fixed: changed string1 != string2 to !(string1.equals(string2))



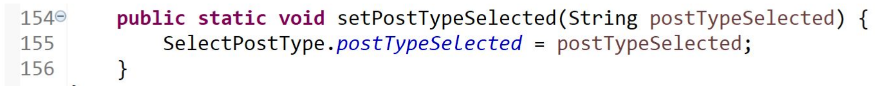
```
47         String[] myRidesLabels = { "Type", "Poster", "Origin", "Destination", "Date", "" };
48         DefaultTableModel model;
49         if (myRides.size() > 0) {
50             myRidesData = new Object[myRides.size()][myRidesLabels.length];
51             for (int r = 0; r < myRides.size(); r++) {
52                 if (!myRides.get(r).getPoster().equalsIgnoreCase(UserService.getInstance().getCurrentUser().getUsername())) {
53                     for (int c = 0; c < 6; c++) {
54                         if (c == 0) {
55                             if (myRides.get(r) instanceof Driver)
56                                 myRidesData[r][c] = new String("Driver");
57                             else
58                                 myRidesData[r][c] = new String("Rider");
59                         } else if (c == 1) {
60                             myRidesData[r][c] = new String(myRides.get(r).getPoster());
61                         } else if (c == 2) {
62                             myRidesData[r][c] = new String(myRides.get(r).getOrigin());
63                         } else if (c == 3) {
64                             myRidesData[r][c] = new String(myRides.get(r).getDest());
65                         } else if (c == 4) {
66                             SimpleDateFormat df = new SimpleDateFormat("MMM dd, yyyy hh:mm a");
67                             String str = df.format(myRides.get(r).getDate());
68                             myRidesData[r][c] = new String(str);
69                         } else if (c == 5) {
70                             myRidesData[r][c] = new String(String.valueOf((myRides.get(r).getID())));
71                         }
72                     }
73                 }
74             }
```

- SelectPostType:
    - Line 155: Write to static field presentation.application.SelectPostType.postTypeSelected from instance method presentation.application.SelectPostType.setPostTypeSelected(String) [Of Concern(15), High confidence]



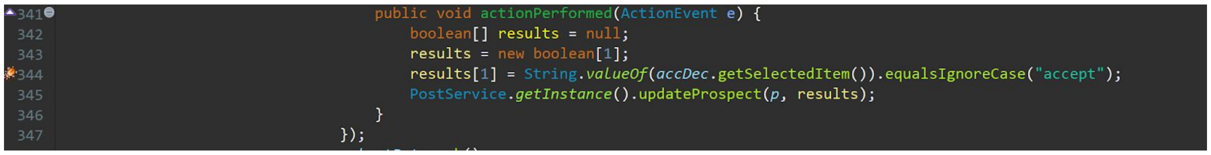
```
154    public void setPostTypeSelected(String postTypeSelected) {
155        this.postTypeSelected = postTypeSelected;
156    }
```

        - Fixed: changed function to be a static function and this to be selectPostType.



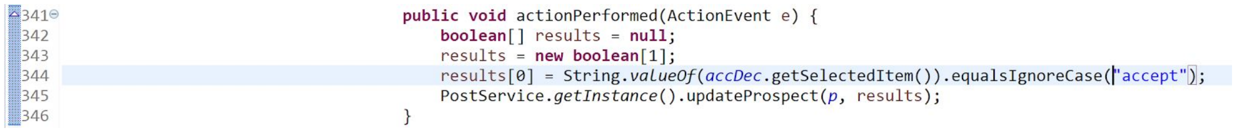
```
154    public static void setPostTypeSelected(String postTypeSelected) {
155        SelectPostType.postTypeSelected = postTypeSelected;
156    }
```

- ViewProfile:
    - Line 344: Array index is out of bounds: 1 [Scary(7), Normal confidence]



```
341                        public void actionPerformed(ActionEvent e) {
342                            boolean[] results = null;
343                            results = new boolean[1];
344                            results[1] = String.valueOf(accDec.getSelectedItem()).equalsIgnoreCase("accept");
345                            PostService.getInstance().updateProspect(p, results);
346                        }
347                    });
```

        - Fixed: changed results[1] to results[0]



```
341                        public void actionPerformed(ActionEvent e) {
342                            boolean[] results = null;
343                            results = new boolean[1];
344                            results[0] = String.valueOf(accDec.getSelectedItem()).equalsIgnoreCase("accept");
345                            PostService.getInstance().updateProspect(p, results);
346                        }
```

Git Analysis

Andrew Ammentorp	- 137 commits
Joseph Perez	- 186 commits
Joseph Yu	- 80 commits
Joshua Huertas	- 137 commits
Leighton Glim	- 285 commits

TimeCards Report

Andrew Ammentorp	- 109 hrs
Joseph Perez	- 104 hrs
Joseph Yu	- 98 hrs
Joshua Huertas	- 98 hrs
Leighton Glim	- 114 hrs

Point Distribution

Andrew Ammentorp	- 1/5th total points
Joseph Perez	- 1/5th total points
Joseph Yu	- 1/5th total points
Joshua Huertas	- 1/5th total points
Leighton Glim	- 1/5th total points

Links

User Guide:
https://github.com/BURS-co/CSI3471-Ride-Share/blob/master/Iteration3/User%20Guide.pdf

Java Docs:
https://github.com/BURS-co/CSI3471-Ride-Share/tree/master/Iteration3/Javadocs

Website:
https://burs-co.github.io/CSI3471-Ride-Share/index.html