M7 Iteration Solid and Grasp Principles

SOLID

S - Single Responsibility Principle

We demonstrate SRP here because our Marketplace class is only responsible for the buy/sell aspect of the game. There is only one reason for a change: if we want to modify transactions in the marketplace. This improves usability and makes the code easy to understand/maintain.

```
    }

    public void buy(Good good) {
        if (good.getBuyPrice() > Main.getUser().getCreditsValue()) {
            control.alertMessage( a: "PURCHASE ERROR",  b: "PRICE OF GOOD EXCEEDS AVAILABLE CREDITS");
        }
        if (Main.getUser().getShip().getCargoCapacity()
                == Main.getUser().getShip().getItemInventory().size()) {
            control.alertMessage( a: "PURCHASE ERROR",  b: "SHIP HAS FULL CARGO CAPACITY");
        }
        Main.getUser().getShip().addToInventory(good);
    }

    public void sell(Good good) {
        if (Main.getUser().getShip().getItemInventory().size() == 0) {
            control.alertMessage( a: "PURCHASE ERROR",  b: "SHIP HAS NO GOODS TO SELL");
        }
        Main.getUser().getShip().removeFromInventory(good);
    }

    public String toString() {
        String str = "";
        for (Good good: goodsAvailable) {
            str += "[" + good.toString() + "], " + "\n";
        }
        return str;
    }

}
```

O - Open/Closed Principle

Our code is open for extension but closed for modification. The Bandit, Trader, and Police class all have startNPC() methods. The startNPCEncounter() method calls the method of the NPC passed in, so it is simple to add NPC encounters without having to modify the code. This is important because we can never break the core of the system.

```
public void startNPCEncounter(NPCencounter npc) {
    npc.startNPC();
}
```

Trader class

```
@Override
Region startNPC() {
    Trader stage = null;
    if ((this).getScene() != null) {
        stage.setScene((this).getScene());
    } else {
        Scene scene = new Scene((this).getTraderPane());
        (this).setScene(scene);
        stage.setScene(scene);
    }

    var rep = new Object() {
        Region resultRegion;
    };
    Region b = null;
    rep.resultRegion = b;

    (this).getBuy().setOnAction(e -> {
        ((this).setBuyChoice((Good)
                (this).getTraderGoodsAvailableDropDown()
                        .getSelectionModel().getSelectedItem());
        (this).playBuy((this).getBuyChoice());
        control.alertMessage( a: "TRADER ALERT",  b: "SUCCESSFUL PURCHASE! ITEM ADDED TO SHIP.");
        stage.setScene(b.getScene());
    });
    (this).getNegotiate().setOnAction(e -> {
        (this).playNegotiate();
        if (Main.getUser().getMerchant() < 5) {
            control.alertMessage( a: "TRADER ALERT",  b: "UNSUCCESSFUL NEGOTIATION :( ");
        } else {
            control.alertMessage( a: "TRADER ALERT",  b: "SUCCESSFUL NEGOTIATION :) ");
        }
        (this).getTraderPane().getChildren().remove((this).getNegotiate());
    });

    (this).getRob().setOnAction(e -> {
        (this).playRob();
        if (Main.getUser().getFighter() < 5) {
            control.alertMessage( a: "TRADER ALERT",  b: "UNSUCCESSFUL ROBBERY :( \n "
                    + "SHIP HEALTH DECREASED TO: "
                    + Main.getUser().getShip().getHealth());
        } else {
            control.alertMessage( a: "TRADER ALERT",  b: "SUCCESSFUL ROBBERY :) ");
        }
        stage.setScene(b.getScene());
    });
    (this).getIgnore().setOnAction(e -> {
        control.alertMessage( a: "TRADER ALERT",  b: "TRADER IGNORED");
        stage.setScene(b.getScene());
    });
    return rep.resultRegion;
}
```

Police class

```
@Override
Region startNPC() {
    Police stage = null;
    if ((this).getScene() != null) {
        stage.setScene(((this).getScene());
    } else {
        Scene scene = new Scene((this).policeInteractPane());
        (this).setScene(scene);
        stage.setScene(scene);
    }

    var rep = new Object() {
        Region resultRegion;
    };
    Region b = null;
    rep.resultRegion = b;

    Controller control = null;
    (this).getForfeit().setOnAction(e -> {
        (this).playForfeit();
        control.alertMessage( a: "POLICE ALERT",  b: "FORFEIT IT IS :( ");
        stage.setScene(b.getScene());
    });

    (this).getFlee().setOnAction(e -> {
        (this).playFlee();
        if (Main.getUser().getPilot() > 5) {
            control.alertMessage( a: "POLICE ALERT",  b: "SUCCESSFUL FLEE :) \n "
                    + "FUEL CAPACITY DECREASED TO: "
                    + Main.getUser().getShip().getFuelCapacity());
        } else {
            control.alertMessage( a: "POLICE ALERT",  b: "UNSUCCESSFUL FLEE :( \n "
                    + "SHIP HEALTH DECREASED TO: "
                    + Main.getUser().getShip().getHealth());
        }
        Region a = null;
        rep.resultRegion = a;
        stage.setScene(a.getScene());
    });

    (this).getFight().setOnAction(e -> {
        (this).playFight();
        if (Main.getUser().getFighter() < 5) {
            control.alertMessage( a: "POLICE ALERT",  b: "UNSUCCESSFUL FIGHT :( ");
        } else {
            control.alertMessage( a: "POLICE ALERT",  b: "SUCCESSFUL FIGHT :) ");
        }
        stage.setScene(b.getScene());

    });
    return b;
}
```

I - Interface Segregation

The Commerce interface has a buy method that both the Trader and Marketplace class implement. Then, branching off from that are two more specific commerce interfaces called TraderCommerce and MarketPlaceCommerce. Both these interfaces have their own methods like traderCommerce has a playBuy(Good good) and MarketPlaceCommerce has sell(Good good). Both the Trader class and MarketPlace class have commerce characteristics which is why they both implement the CommerceInterface, but they also have their own individual commerce characteristics which is why they also implement their own respective commerce interface. The Trader class implements the TraderCommerce interface and uses the playBuy method that is a characteristic of only that class. Likewise, the MarketPlace class implements the MarketPlaceCommerce interface which has a method that is important to the selling attribute of the Marketplace class (not needed in Trader class). This is the implementation of interface segregation as there is a large interface that holds all the methods that both Trader and Marketplace would use, and then separate interfaces that hold methods that are unique to their respective classes.

```java
public interface TraderCommerce {
    public abstract void playBuy(Good good);
}
```

```java
public interface CommerceInterface {
    public abstract void buy(Good good);

}
```

```java
public class Trader implements CommerceInterface, TraderCommerce {
```

User.java | Region.java | Bandit.java | Police.java | Trader.java | TraderInterface.java

```java
        //GOODS THAT TRADER SELLS
        ObservableList<Good> traderGoodsList = FXCollections.observableArrayList();
        for (Good goods : traderInventory) {
            traderGoodsList.add(goods);

        }
        traderGoodsAvailableDropDown.setItems(traderGoodsList);
        traderPane.add(traderGoodsAvailableDropDown, i: 1, i1: 0);
        return traderPane;
    }

    public void playBuy(Good good) {
        buy(good);
        Main.getUser().getShip().addToInventory(good);
    }

    public void buy(Good good) {
        if (good.getBuyPrice() > Main.getUser().getCreditsValue()) {
            control.alertMessage( a: "PURCHASE ERROR", b: "PRICE OF GOOD EXCEEDS AVAILABLE CREDITS
        }
        if (Main.getUser().getShip().getCargoCapacity()
                == Main.getUser().getShip().getItemInventory().size()) {
            control.alertMessage( a: "PURCHASE ERROR", b: "SHIP HAS FULL CARGO CAPACITY");
        }
    }
```

```java
package sample;

public interface MarketPlaceCommerce {
    public abstract void sell(Good good);
}
```

```java
public class Marketplace implements CommerceInterface, MarketPlaceCommerce {
    private String name;
    private ArrayList<Good> goodsAvailable = new ArrayList<>();
```

va × Ship.java × User.java × Region.java × Trader.java × Bandit.java × Police.java × Marketplace.java ×

```java
        if (a <= 7) {
            goodsAvailable.add(new Good( name: "Gold Tinted Sword", techLevel,  basePrice: 3));
        }
    }

    public void buy(Good good) {
        if (good.getBuyPrice() > Main.getUser().getCreditsValue()) {
            control.alertMessage( a: "PURCHASE ERROR",  b: "PRICE OF GOOD EXCEEDS AVAILABLE CREDITS");
        }
        if (Main.getUser().getShip().getCargoCapacity()
                == Main.getUser().getShip().getItemInventory().size()) {
            control.alertMessage( a: "PURCHASE ERROR",  b: "SHIP HAS FULL CARGO CAPACITY");
        }
        if (good.getName().equals("5 Units of Health")) {
            Main.getUser().getShip().setHealth(Main.getUser().getShip().getHealth() + 5);
        } else if (good.getName().equals("5 Units of Ship Fuel")) {
            Main.getUser().getShip().setFuelCapacity(Main.getUser().getShip().getFuelCapacity() + 5);
        } else {
            Main.getUser().getShip().addToInventory(good);
        }
    }

    public void sell(Good good) {
        if (Main.getUser().getShip().getItemInventory().size() == 0) {
            control.alertMessage( a: "PURCHASE ERROR",  b: "SHIP HAS NO GOODS TO SELL");
        }
        Main.getUser().getShip().removeFromInventory(good);
    }
```

GRASP

I - Information Expert

The universe class has the tech levels, coordinate, and region descriptions information, therefore we decided to implement the generateRegion method in the Universe class. The generaterRegion method requires this data to complete the functionality. This results in a better organization of code and clear functionality.

```java
package sample;

import javafx.scene.layout.GridPane;

public interface AreaInt {
    public Region generateRegion();
    public GridPane createPane();
}
```

```java
public class Universe implements AreaInt {
    private Region[] regionArr = new Region[10];
    private String[] nameArr =  {"Venus", "Earth", "Mars", "Neptune", "Mercury",
        "Saturn", "Jupiter", "Pluto", "Zeon", "Uranus"
    };
    private int[] techLevels =  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
        11, 12, 13, 14, 15, 16, 17, 18, 19, 20
    };
    private int[][] coordinates =  {{0, 0}, {1, 0}, {2, 0}, {3, 0},
        {0, 1}, {1, 1}, {2, 1}, {3, 1}, {0, 2}, {1, 2}
    };
    private String[] descriptions = {"This region is filled with mountains, "
            + "plains, high plateaus, canyons, volcanoes, and ridges. "
            + "It has a temperature of 864 degrees Fahrenheit. "
            + "Local inhabitants of this region are called the Venetians "
            + "and don't interact with space invaders that well.", "This region "
            + "is filled with mountains, oceans, and prairies."
            + "It has a temperature of 70 degrees Fahrenheit. Local inhabitants of this region are "
            + "called the Humans and don't interact with space invaders that well.", "This "
            + "region is filled with dark slope streaks, dust devil tracks, and rivers. "
                + "It has a temperature of -125 degrees C. "
                + "Local inhabitants of this region are called the Martians "
            + "and don't interact with space invaders that well. ", "This "
            + "region is very blue and filled with icy slush and dust. "
                + "It has a temperature of -392 degrees F. "
                + "Local inhabitants of this region are called the Neptunians"
                + " and don't interact with space invaders that well. ", "This "
            + "region is filled with lunar highlands and cliffs. "
                + "It has a temperature of 801 degrees F. "
                + "Local inhabitants of this region are called the "
                + "Mercurians and don't interact with space invaders that well. ", "This "
            + "region is filled with swirling gases and liquids deeper down. "
                + "It has a temperature of -178 degrees Celcius. "
                + "Local inhabitants of this region are called the "
                + "Sundials and don't interact with space invaders that well. ", "This "
            + "region is filled with grassy fields and dirt pathways."
                + " It has a temperature of -578 degrees Celcius. "
                + "Local inhabitants of this region are called the Loners "
                + "and don't interact with space invaders that well. ", "This "
            + "region is filled with cliffs, caves, and lakes. "
                + "It has a temperature of -790 degrees Celcius. "
                + "Local inhabitants of this region are called the Carls"
                + " and don't interact with space invaders that well. ", "This "
            + "region is filled with dark slope streaks, salt lakes, "
                + "icy glaciers, and rivers. It has a temperature of 854 degrees F. "
                + "Local inhabitants of this region are called the "
                + "Amomas and don't interact with space invaders that well. ", "This "
            + "region is full of emptiness. It is very large, but cold and windy."
                + "It has a temperature of -1000 degrees F. "
                + "Local inhabitanats of this region is called Uranians."
    };
```

```java
    public Region generateRegion() {
        Random random = new Random();

        String name = null;
        int nameIndex = random.nextInt( bound: 10);
        while (name == null) {
            if (nameArr[nameIndex] != null) {
                name = nameArr[nameIndex];
            } else {
                nameIndex = random.nextInt( bound: 10);
            }
        }
        nameArr[nameIndex] = null;

        int techLevel = -1;
        int techIndex = random.nextInt( bound: 20);
        while (techLevel == -1) {
            if (techLevels[techIndex] != -1) {
                techLevel = techLevels[techIndex];
            } else {
                techIndex = random.nextInt( bound: 20);
            }
        }
        techLevels[techIndex] = -1;

        int[] coordinate = null;
        int coordinateIndex = random.nextInt( bound: 10);
        while (coordinate == null) {
            if (coordinates[coordinateIndex] != null) {
                coordinate = coordinates[coordinateIndex];
            } else {
                coordinateIndex = random.nextInt( bound: 10);
            }
        }

        coordinates[coordinateIndex] = null;
        String description = descriptions[nameIndex];
        return new Region(techLevel, coordinate, name, description);
    }
```

C - Controller

The controller is the first object past the UI that processes the system operation for alert messages. This ensures that the UI does not have responsibility for fulfilling system events. In this example, the controller does not need to delegate tasks to another class.

```
package sample;

import javafx.scene.control.Alert;

public class Controller {
    public void alertMessage(String a, String b) {
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle(a);
        alert.setHeaderText(null);
        alert.setContentText(b);
        alert.showAndWait();
    }
}
```

```
if (user.getUsername() == null || user.getUsername().equals("")) {
    control.alertMessage( a: "USERNAME ERROR",  b: "USERNAME CANNOT BE EMPTY");
    return;
} else if (user.getDifficultyChoice() == null) {
    control.alertMessage( a: "DIFFICULTY CHOICE ERROR",
            b: "MUST SELECT A DIFFICULTY LEVEL FROM DROPDOWN MENU");
    return;
} else if (fighterField.getText().equals("") || pilotField.getText().equals("")
        || merchantField.getText().equals("") || engineerField.getText().equals("")) {
    control.alertMessage( a: "INPUT ERROR",
            b: "ALL THE SKILL POINTS SHOULD BE FILLED OUT");
}
if (user.getDifficultyChoice().equals("EASY (15pt)")) {
    user.setDifficultyPoints(15);
    user.setCreditsValue(1000);
} else if (user.getDifficultyChoice().equals("MEDIUM (10pt)")) {
    user.setDifficultyPoints(10);
    user.setCreditsValue(500);
} else if (user.getDifficultyChoice().equals("HARD (5pt)")) {
    user.setDifficultyPoints(5);
    user.setCreditsValue(100);
}
if (user.getEngineer() < 0 || user.getFighter() < 0
        || user.getMerchant() < 0 || user.getPilot() < 0) {
    control.alertMessage( a: "SKILL POINTS ERROR",
            b: "SKILL POINTS CANNOT BE NEGATIVE");
    return;
} else if (user.skillPointSum() > user.getDifficultyPoints()) {
    control.alertMessage( a: "SKILL POINTS ERROR",
            b: "NOT ENOUGH POINTS TO ALLOCATE: YOU ONLY HAVE "
                    + user.getDifficultyPoints());
    return;
```

P - Polymorphism

The example below shows an example of Polymorphism used in our code. The Bandit, Police and Trader classes are all instances of NPC encounters, however, they have their own functionality. This is useful for when we want to handle new variations.

```
package sample;

import javafx.scene.Scene;

public abstract class NPCencounter {
    abstract Scene getScene();


}
```

```
public class Bandit extends NPCencounter {
    private Button payDemand;
```

```
public class Police extends NPCencounter {
```

```
public class Trader extends NPCencounter {
```

L - Low Coupling

In this example our main class calls the region class, the region class calls the marketplace class, and the marketplace class implements the functionality. This shows low coupling because each class has its own responsibility. This minimizes the dependency which makes the system maintainable and efficient.

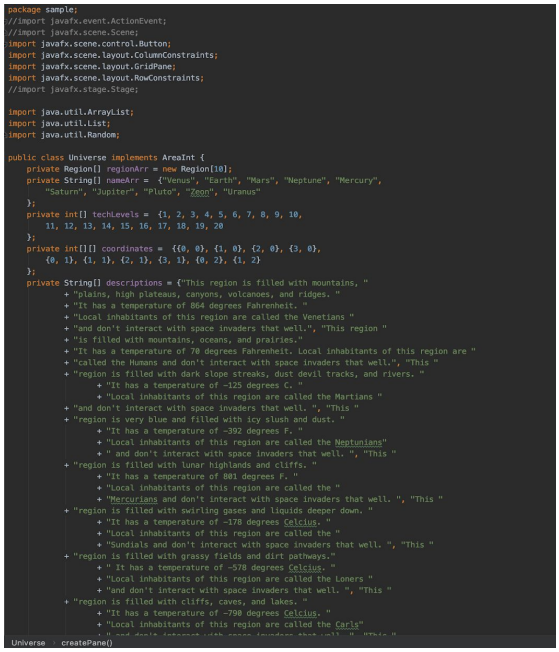
```
Label goodsAvailableLabel = new Label( s: "Goods Available To Buy:");
regionPane.add(goodsAvailableLabel, i: 0, i1: 1);
ObservableList<Good> goodsList = FXCollections.observableArrayList();
for (Good good: getMarket().getGoodsAvailable()) {
    goodsList.add(good);
}
goodsAvailableDropDown.setItems(goodsList);
regionPane.add(goodsAvailableDropDown, i: 1, i1: 1);
money = new Label(creditDisplay);
regionPane.add(money, i: 0, i1: 2);
Label shipContentLabel = new Label( s: "Goods Available To Sell:");
regionPane.add(shipContentLabel, i: 2, i1: 1);
regionPane.add(Main.getUser().getShip().shipContentDropDown(), i: 3, i1: 1);

map = new Button( s: "map");
update = new Button( s: "update");
regionPane.add(map, i: 0, i1: 3);
regionPane.add(market.getBuy(), i: 1, i1: 3);
regionPane.add(market.getSell(), i: 2, i1: 3);
regionPane.add(update, i: 3, i1: 3);

setVisited(true);
setRegionPane(regionPane);
return regionPane;
```

```
private void startRandomRegion() {
    var ref = new Object() {
        Scene universeScene;
        GridPane universePane = universe.createPane();
        int regionIndex = random.nextInt( bound: 10);
        Region randomRegion;
        Scene randomScene;
    };
    Button ranRegionBtn = universe.getArrButtons()[ref.regionIndex];
    ref.universeScene = new Scene(ref.universePane);

    ref.randomRegion = universe.getRegionArr()[ref.regionIndex];
    ref.randomScene = new Scene(ref.randomRegion.createGridPane());
    ref.randomRegion.setScene(ref.randomScene);
    stage.setScene(ref.randomRegion.getScene());

    ref.randomRegion.getUpdate().setOnAction(e -> {
        ref.randomRegion.updateShipDropDown();
        ref.randomRegion.updateMoneyLabel();
    });

    ref.randomRegion.getMarket().getBuy().setOnAction(e -> {
        ref.randomRegion.setBuyChoice((Good)
                ref.randomRegion.getGoodsAvailableDropDown()
                        .getSelectionModel().getSelectedItem());
        ref.randomRegion.getMarket().buy(ref.randomRegion.getBuyChoice());
    });

    ref.randomRegion.getMarket().getSell().setOnAction(e -> {
        ref.randomRegion.setSellChoice();
        ref.randomRegion.getMarket().sell(ref.randomRegion.getBuyChoice());
    });

    ranRegionBtn.setTooltip(new Tooltip( s: ref.randomRegion.getName()
            + "\n" + ref.randomRegion.getDescription() + "\n"
            + ref.randomRegion.getTechLevel()));
    //RANDOM REGION --> MAP
    ref.randomRegion.getMapButton().setOnAction(e -> {
        stage.setScene(ref.universeScene);
    });
    startUniverse(ref.universeScene, ref.randomRegion);
}
```

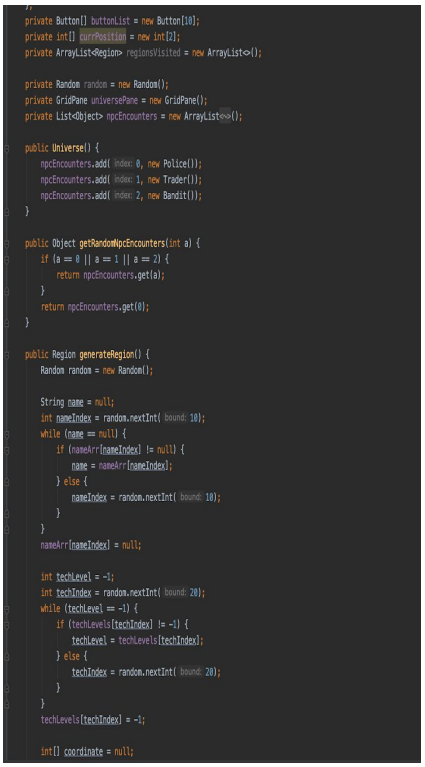
```
public void buy(Good good) {
    if (good.getBuyPrice() > Main.getUser().getCreditsValue()) {
        control.alertMessage( a: "PURCHASE ERROR", b: "PRICE OF GOOD EXCEEDS AVAILABLE CREDITS");
    }
    if (Main.getUser().getShip().getCargoCapacity()
            == Main.getUser().getShip().getItemInventory().size()) {
        control.alertMessage( a: "PURCHASE ERROR", b: "SHIP HAS FULL CARGO CAPACITY");
    }
    Main.getUser().getShip().addToInventory(good);
}

public void sell(Good good) {
    if (Main.getUser().getShip().getItemInventory().size() == 0) {
        control.alertMessage( a: "PURCHASE ERROR", b: "SHIP HAS NO GOODS TO SELL");
    }
    Main.getUser().getShip().removeFromInventory(good);
}
```

P - Pure Fabrication

The universe class does not represent a domain object. However, it performs all generate region and calculate distance features. This provides high cohesion, low coupling and the ability for reuse.



```
package sample;
//import javafx.event.ActionEvent;
//import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.ColumnConstraints;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.RowConstraints;
//import javafx.stage.Stage;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class Universe implements AreaInt {
    private Region[] regionArr = new Region[10];
    private String[] nameArr =  {"Venus", "Earth", "Mars", "Neptune", "Mercury",
        "Saturn", "Jupiter", "Pluto", "Zeon", "Uranus"
    };
    private int[] techLevels =  {1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
        11, 12, 13, 14, 15, 16, 17, 18, 19, 20
    };
    private int[][] coordinates =  {{0, 0}, {1, 0}, {2, 0}, {3, 0},
        {0, 1}, {1, 1}, {2, 1}, {3, 1}, {0, 2}, {1, 2}
    };
    private String[] descriptions = {"This region is filled with mountains, "
            + "plains, high plateaus, canyons, volcanoes, and ridges. "
            + "It has a temperature of 864 degrees Fahrenheit. "
            + "Local inhabitants of this region are called the Venetians "
            + "and don't interact with space invaders that well.", "This region "
            + "is filled with mountains, oceans, and prairies."
            + "It has a temperature of 70 degrees Fahrenheit. Local inhabitants of this region are "
            + "called the Humans and don't interact with space invaders that well.", "This "
            + "region is filled with dark slope streaks, dust devil tracks, and rivers. "
                + "It has a temperature of -125 degrees C. "
                + "Local inhabitants of this region are called the Martians "
            + "and don't interact with space invaders that well. ", "This "
            + "region is very blue and filled with icy slush and dust. "
                + "It has a temperature of -392 degrees F. "
                + "Local inhabitants of this region are called the Neptunians"
                + " and don't interact with space invaders that well. ", "This "
            + "region is filled with lunar highlands and cliffs. "
                + "It has a temperature of 801 degrees F. "
                + "Local inhabitants of this region are called the "
                + "Mercurians and don't interact with space invaders that well. ", "This "
            + "region is filled with swirling gases and liquids deeper down. "
                + "It has a temperature of -178 degrees Celcius. "
                + "Local inhabitants of this region are called the "
                + "Sundials and don't interact with space invaders that well. ", "This "
            + "region is filled with grassy fields and dirt pathways."
                + " It has a temperature of -570 degrees Celcius. "
                + "Local inhabitants of this region are called the Loners "
                + "and don't interact with space invaders that well. ", "This "
            + "region is filled with cliffs, caves, and lakes. "
                + "It has a temperature of -790 degrees Celcius. "
                + "Local inhabitants of this region are called the Carls"
Universe  ›  createPane()
```



```
    private Button[] buttonList = new Button[10];
    private int[] currPosition = new int[2];
    private ArrayList<Region> regionsVisited = new ArrayList<>();

    private Random random = new Random();
    private GridPane universePane = new GridPane();
    private List<Object> npcEncounters = new ArrayList<>();

    public Universe() {
        npcEncounters.add( index: 0, new Police());
        npcEncounters.add( index: 1, new Trader());
        npcEncounters.add( index: 2, new Bandit());
    }

    public Object getRandomNpcEncounters(int a) {
        if (a == 0 || a == 1 || a == 2) {
            return npcEncounters.get(a);
        }
        return npcEncounters.get(0);
    }

    public Region generateRegion() {
        Random random = new Random();

        String name = null;
        int nameIndex = random.nextInt( bound: 10);
        while (name == null) {
            if (nameArr[nameIndex] != null) {
                name = nameArr[nameIndex];
            } else {
                nameIndex = random.nextInt( bound: 10);
            }
        }
        nameArr[nameIndex] = null;

        int techLevel = -1;
        int techIndex = random.nextInt( bound: 20);
        while (techLevel == -1) {
            if (techLevels[techIndex] != -1) {
                techLevel = techLevels[techIndex];
            } else {
                techIndex = random.nextInt( bound: 20);
            }
        }
        techLevels[techIndex] = -1;

        int[] coordinate = null;
```

```
    public GridPane createPane() {
        for (int i = 0; i < 4; i++) {
            ColumnConstraints column = new ColumnConstraints( v: 125);
            universePane.getColumnConstraints().add(column);
        }
        for (int i = 0; i < 3; i++) {
            RowConstraints row = new RowConstraints( v: 100);
            universePane.getRowConstraints().add(row);
        }
        for (int i = 0; i < 10; i++) {
            regionArr[i] = generateRegion();
        }
        for (int i = 0; i < 10; i++) {
            Button regionButton;
            if (regionArr[i].isVisited()) {
                regionButton = new Button(regionArr[i].getName());
            } else {
                regionButton = new Button( s: regionArr[i].coordinateToString()
                        + " " + calculateDistance(regionArr[i]));
            }
            buttonList[i] = regionButton;
            regionButton.setMaxSize(Double.MAX_VALUE, Double.MAX_VALUE);
            universePane.add(regionButton,
                    regionArr[i].getCoordinateX(), regionArr[i].getCoordinateY());
        }
        return universePane;
    }

    public double calculateDistance(Region region) {
        int xCoordinate = region.getCoordinateX();
        int yCoordinate = region.getCoordinateY();
        return Math.sqrt(Math.pow((xCoordinate - currPosition[0]), 2)
                + Math.pow(yCoordinate - currPosition[1], 2));
    }

    public Button[] getArrButtons() { return buttonList; }

    public Region[] getRegionArr() { return regionArr; }
}
```