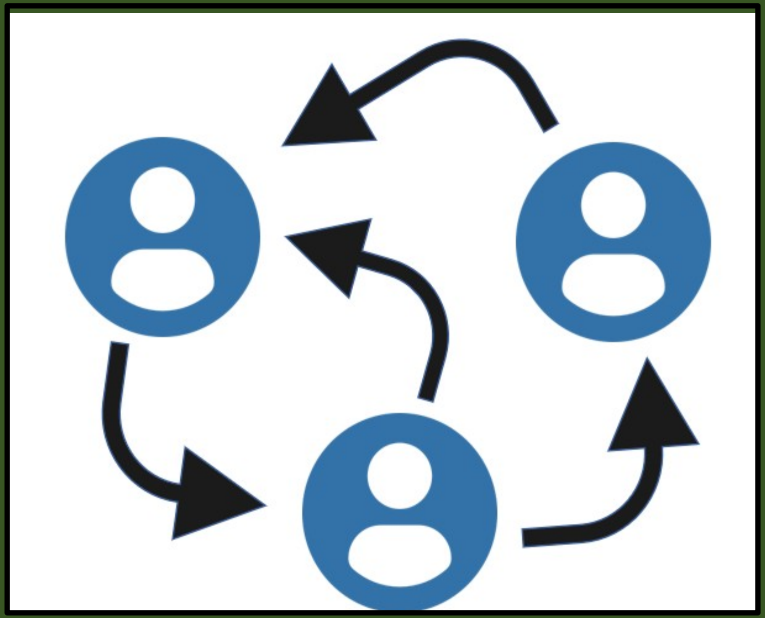
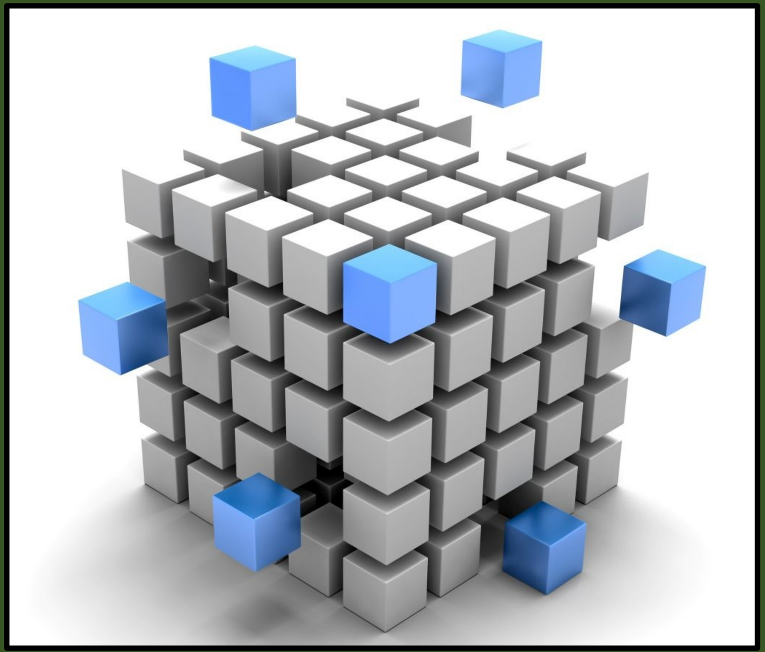
Team T02

May 11th, 2021

Tanner Evans *(team manager)*	Thomas Bowidowicz *(document manager)*	Robin Acosta	Jared Bock	Marcos Lopez	Jacob Varela

System Architecture Design Choices





Actor Model

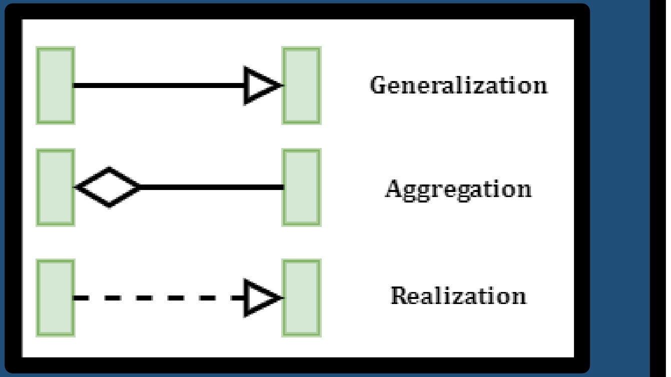
- Actor model sees each actor being functional as its own independent subsystem.
- Actors can send and and receive messages and choose how to respond to the input locally.
- Allows for asynchronous communication.

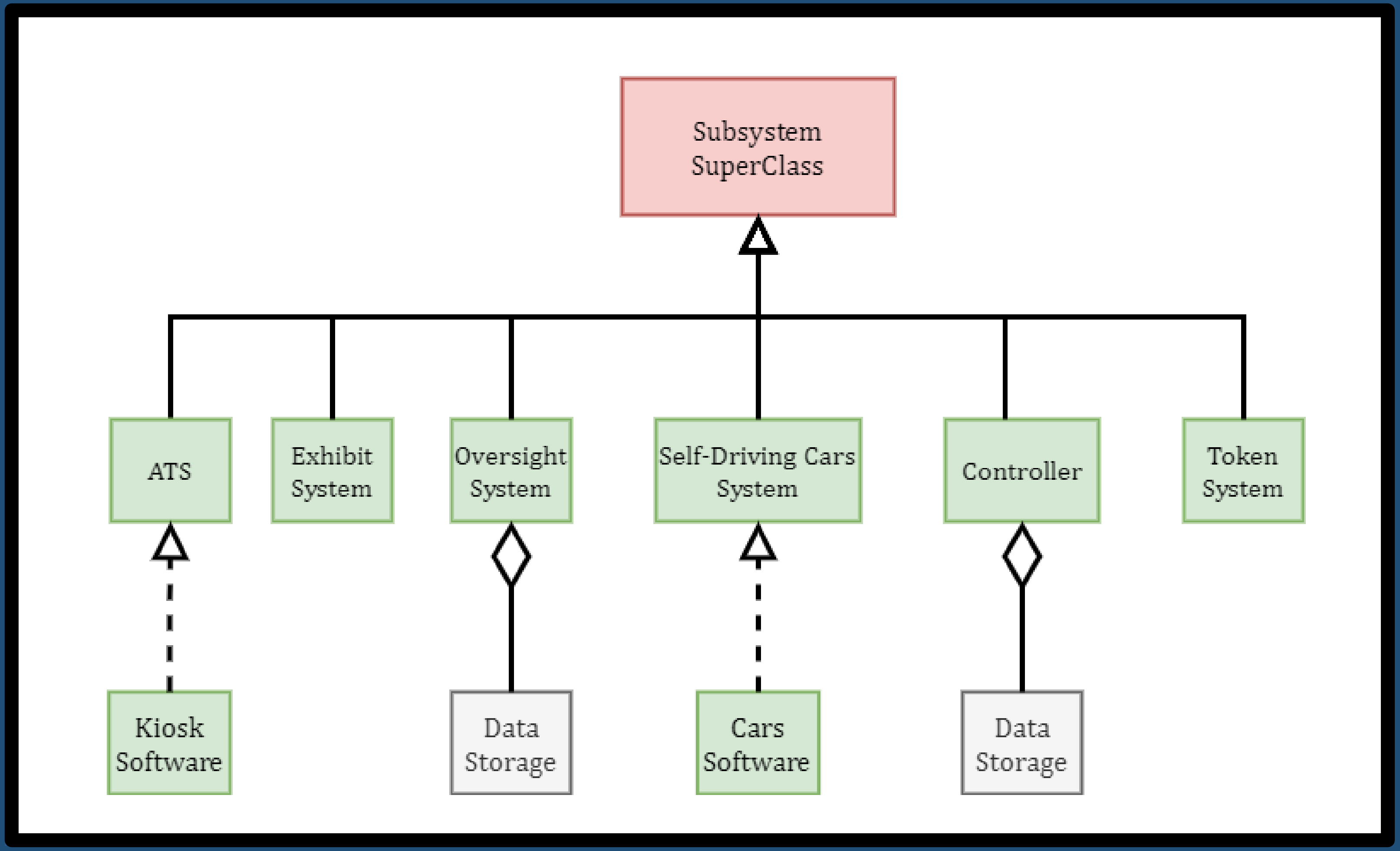
Modular Subclassing

- Superclass can be defined with the majority of the universal functionality while subclasses can define their specific requirements.
- Allows for modularity of design and easy additions or removals of components to the system.
- Superclass Component is extended by all subclasses defines the general outline for all components.

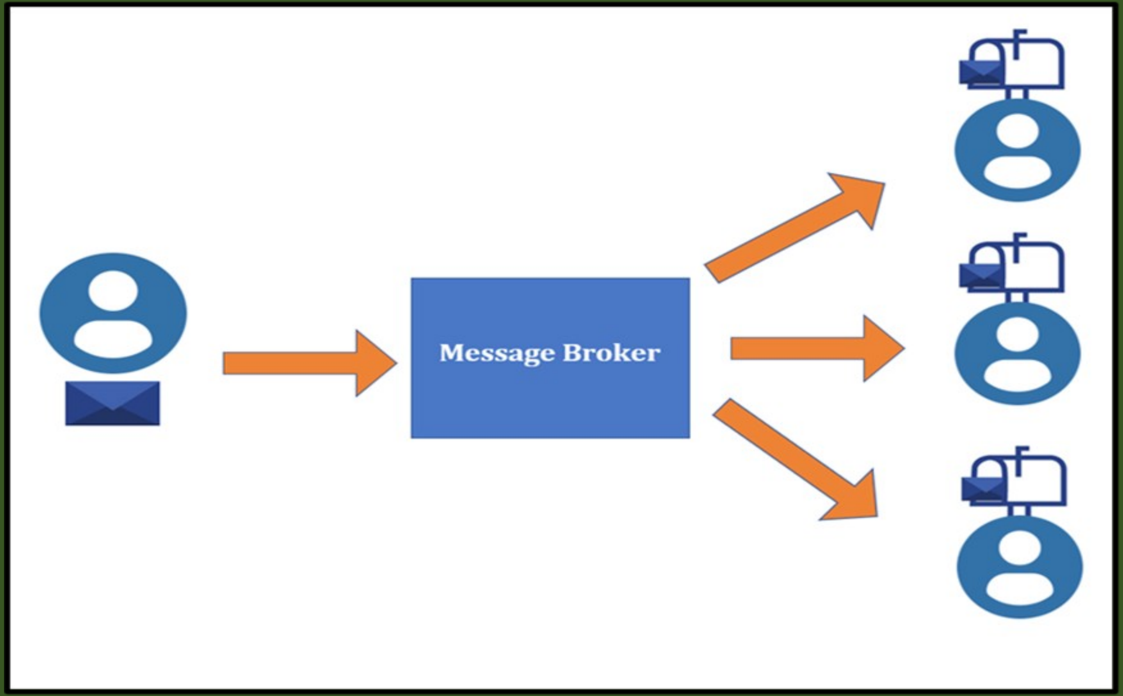
Class Diagram

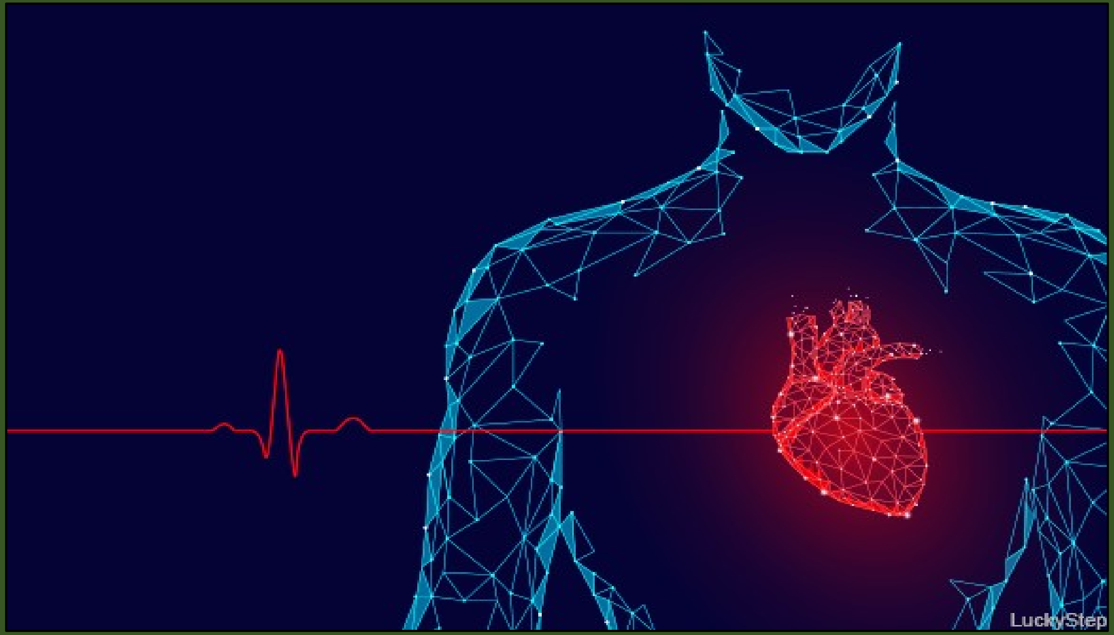
Legend





System Architecture Design Choices *Continued…*





Message Broker

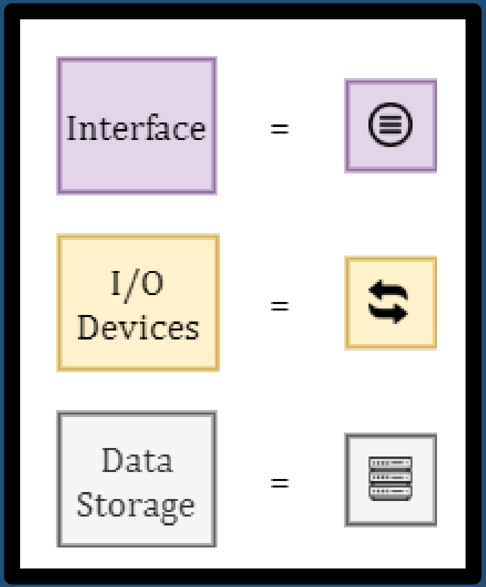
- Message broker mediates various messages from range of components while they have minimal awareness of each other.
- Predefined messages are sent from components to broker and placed on a message queue before being delivered to destination.
- Allows for asynchronous communication between components.

Heartbeat Signals

- Used for notifying controller that devices are operating normally.
- Periodic notification sent to indicate normal function.
- If signal stops, it indicates a malfunction or break in the device.

System Architecture

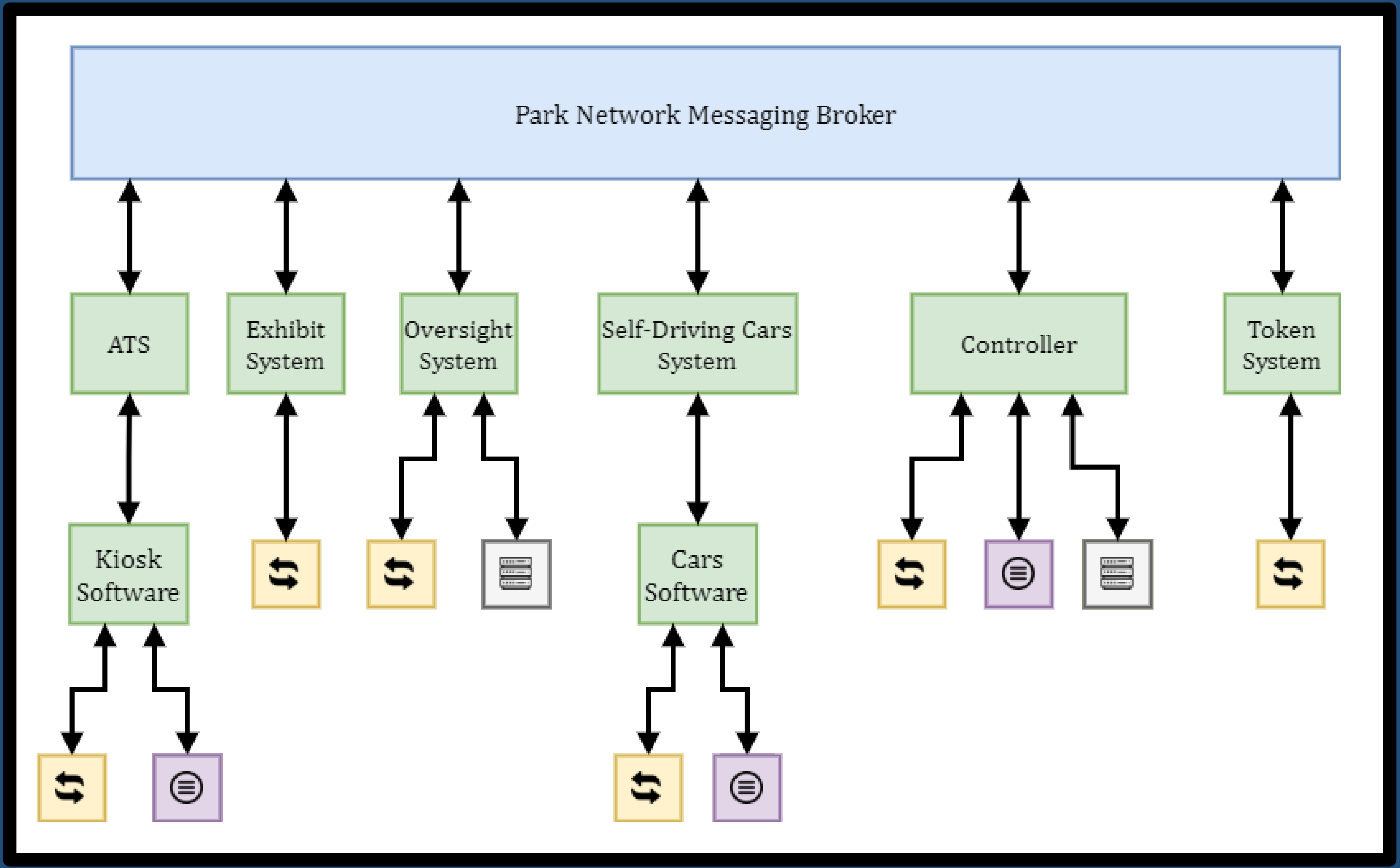
Legend



Interface =

I/O Devices =

Data Storage =



Park Network Messaging Broker

ATS

Exhibit System

Oversight System

Self-Driving Cars System

Controller

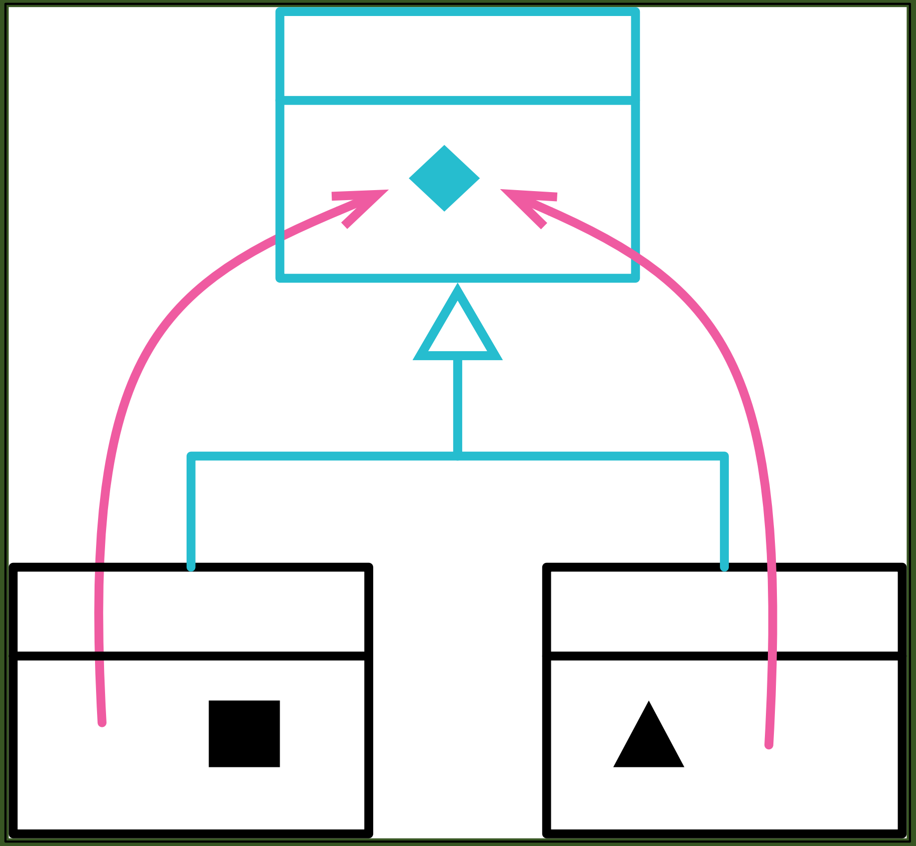
Token System

Kiosk Software

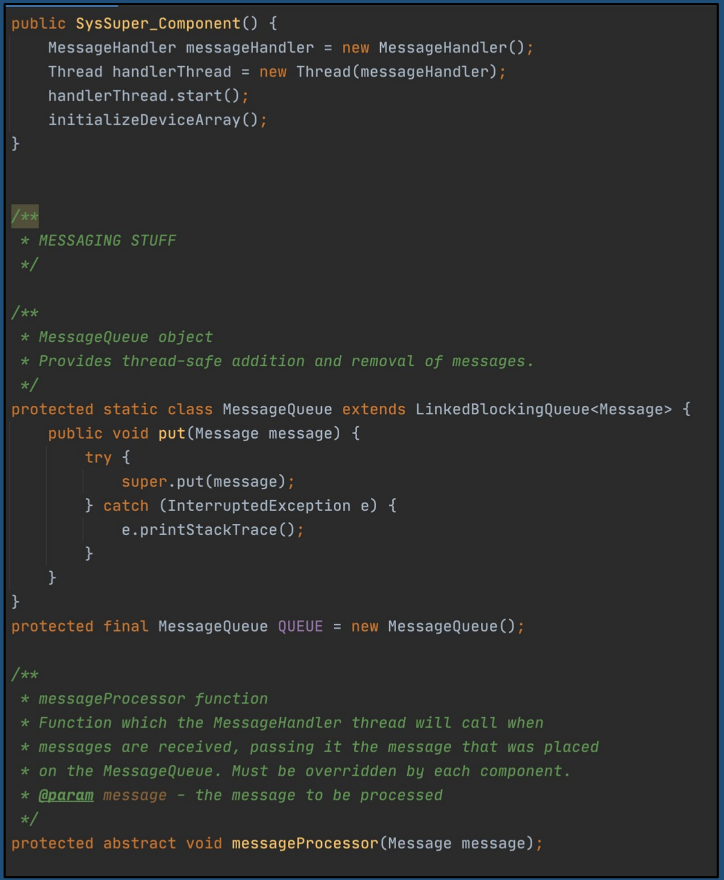
Cars Software

Component Superclass

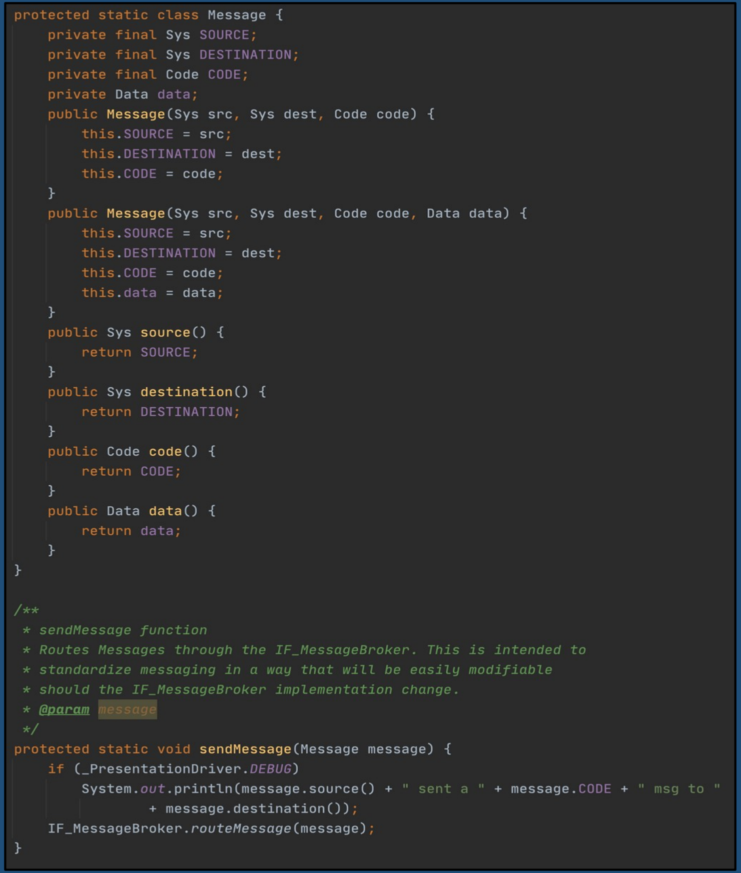
- The Component superclass defines most of the functionality that the various component subsystems share.
- Most of the message functionality is defined in Component class.
- The following core capabilities are defined in the superclass:
  - MessageQueue
  - MessageHandler
  - Message object
  - Stream object
  - SendMessage
  - HeartbeatTimer
  - Data object
  - Device object



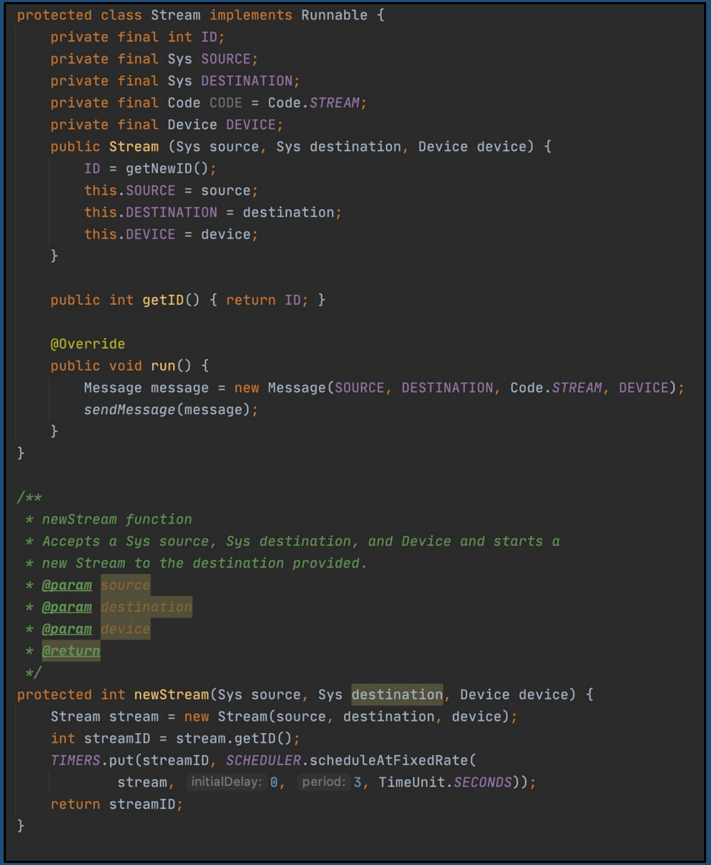
Component Superclass – Code Sample



```java
public SysSuper_Component() {
    MessageHandler messageHandler = new MessageHandler();
    Thread handlerThread = new Thread(messageHandler);
    handlerThread.start();
    initializeDeviceArray();
}


/**
 * MESSAGING STUFF
 */

/**
 * MessageQueue object
 * Provides thread-safe addition and removal of messages.
 */
protected static class MessageQueue extends LinkedBlockingQueue<Message> {
    public void put(Message message) {
        try {
            super.put(message);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
protected final MessageQueue QUEUE = new MessageQueue();

/**
 * messageProcessor function
 * Function which the MessageHandler thread will call when
 * messages are received, passing it the message that was placed
 * on the MessageQueue. Must be overridden by each component.
 * @param message - the message to be processed
 */
protected abstract void messageProcessor(Message message);
```



```java
protected static class Message {
    private final Sys SOURCE;
    private final Sys DESTINATION;
    private final Code CODE;
    private Data data;
    public Message(Sys src, Sys dest, Code code) {
        this.SOURCE = src;
        this.DESTINATION = dest;
        this.CODE = code;
    }
    public Message(Sys src, Sys dest, Code code, Data data) {
        this.SOURCE = src;
        this.DESTINATION = dest;
        this.CODE = code;
        this.data = data;
    }
    public Sys source() {
        return SOURCE;
    }
    public Sys destination() {
        return DESTINATION;
    }
    public Code code() {
        return CODE;
    }
    public Data data() {
        return data;
    }
}

/**
 * sendMessage function
 * Routes Messages through the IF_MessageBroker. This is intended to
 * standardize messaging in a way that will be easily modifiable
 * should the IF_MessageBroker implementation change.
 * @param message
 */
protected static void sendMessage(Message message) {
    if (_PresentationDriver.DEBUG)
        System.out.println(message.source() + " sent a " + message.CODE + " msg to "
                + message.destination());
    IF_MessageBroker.routeMessage(message);
}
```

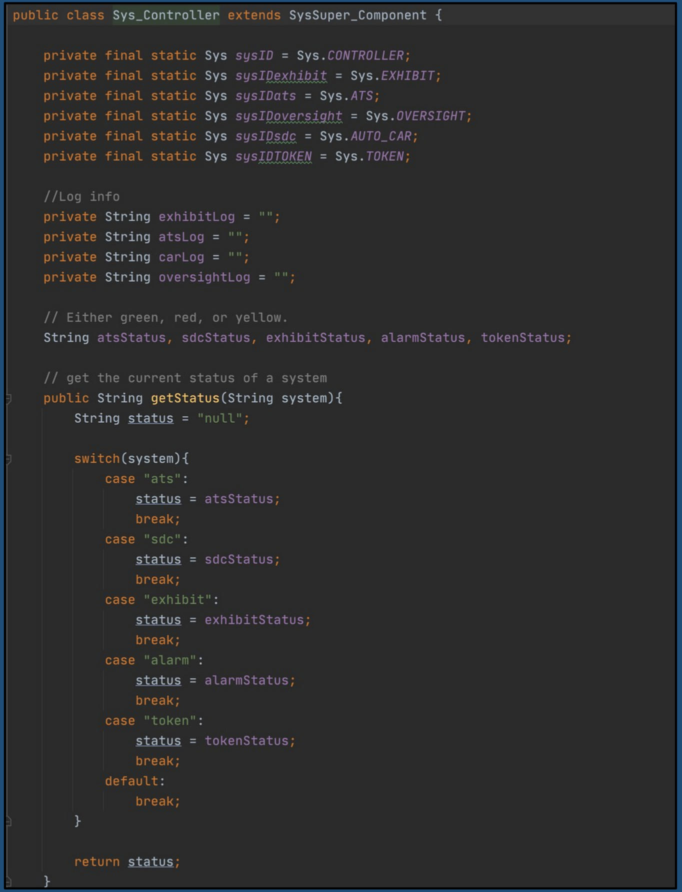


```java
protected class Stream implements Runnable {
    private final int ID;
    private final Sys SOURCE;
    private final Sys DESTINATION;
    private final Code CODE = Code.STREAM;
    private final Device DEVICE;
    public Stream (Sys source, Sys destination, Device device) {
        ID = getNewID();
        this.SOURCE = source;
        this.DESTINATION = destination;
        this.DEVICE = device;
    }

    public int getID() { return ID; }

    @Override
    public void run() {
        Message message = new Message(SOURCE, DESTINATION, Code.STREAM, DEVICE);
        sendMessage(message);
    }
}

/**
 * newStream function
 * Accepts a Sys source, Sys destination, and Device and starts a
 * new Stream to the destination provided.
 * @param source
 * @param destination
 * @param device
 * @return
 */
protected int newStream(Sys source, Sys destination, Device device) {
    Stream stream = new Stream(source, destination, device);
    int streamID = stream.getID();
    TIMERS.put(streamID, SCHEDULER.scheduleAtFixedRate(
            stream,  initialDelay: 0,  period: 3, TimeUnit.SECONDS));
    return streamID;
}
```

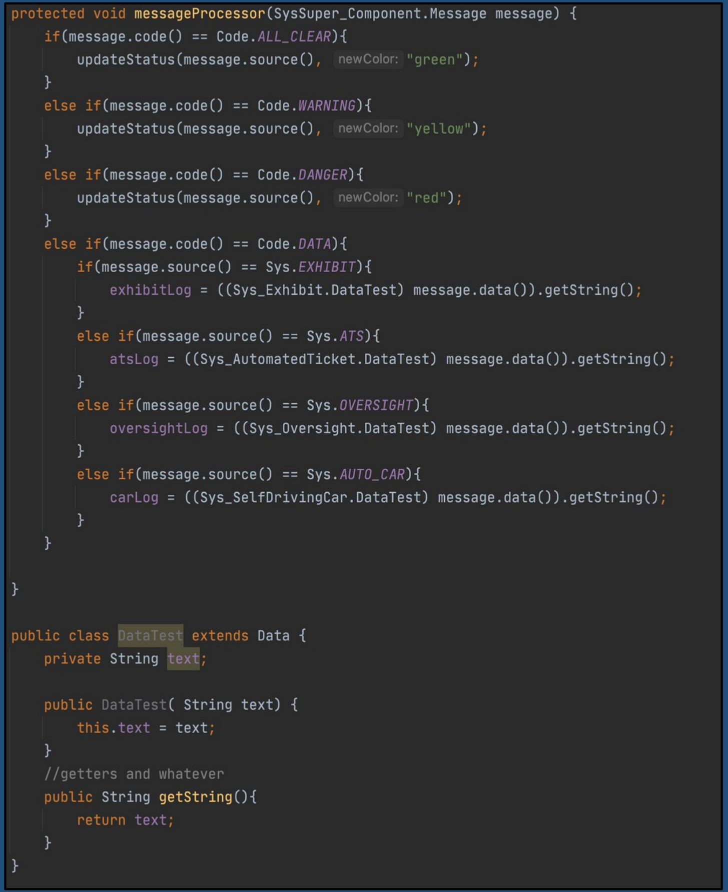
Components – Siesta Gardens Controller (SGC)

- Serves as main controller for park subsystems.
- Is the primary destination for subcomponent messages containing data.
- GUI allows for park administrators to look at current status of park and to respond to any issues.
- Receives messages and data through the message broker from the following subcomponents:
    - Token Monitoring System
    - Oversight System
    - Exhibit System
    - Self Driving Car System
    - Automated Ticket System

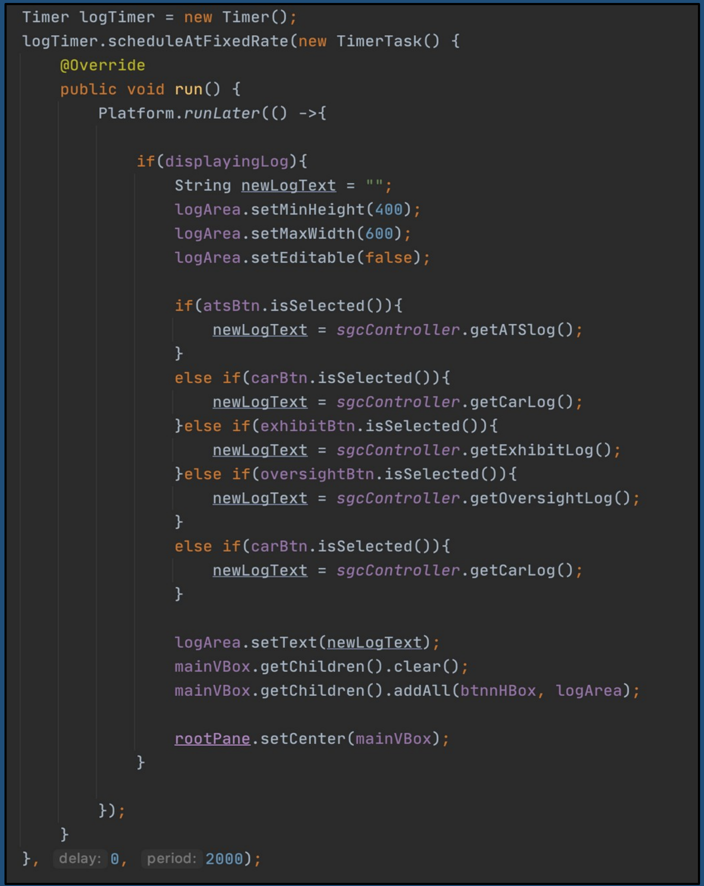
Siesta Gardens Controller – Code Sample



```java
public class Sys_Controller extends SysSuper_Component {

    private final static Sys sysID = Sys.CONTROLLER;
    private final static Sys sysIDexhibit = Sys.EXHIBIT;
    private final static Sys sysIDats = Sys.ATS;
    private final static Sys sysIDoversight = Sys.OVERSIGHT;
    private final static Sys sysIDsdc = Sys.AUTO_CAR;
    private final static Sys sysIDTOKEN = Sys.TOKEN;

    //Log info
    private String exhibitLog = "";
    private String atsLog = "";
    private String carLog = "";
    private String oversightLog = "";

    // Either green, red, or yellow.
    String atsStatus, sdcStatus, exhibitStatus, alarmStatus, tokenStatus;

    // get the current status of a system
    public String getStatus(String system){
        String status = "null";

        switch(system){
            case "ats":
                status = atsStatus;
                break;
            case "sdc":
                status = sdcStatus;
                break;
            case "exhibit":
                status = exhibitStatus;
                break;
            case "alarm":
                status = alarmStatus;
                break;
            case "token":
                status = tokenStatus;
                break;
            default:
                break;
        }

        return status;
    }
```



```java
protected void messageProcessor(SysSuper_Component.Message message) {
    if(message.code() == Code.ALL_CLEAR){
        updateStatus(message.source(), newColor: "green");
    }
    else if(message.code() == Code.WARNING){
        updateStatus(message.source(), newColor: "yellow");
    }
    else if(message.code() == Code.DANGER){
        updateStatus(message.source(), newColor: "red");
    }
    else if(message.code() == Code.DATA){
        if(message.source() == Sys.EXHIBIT){
            exhibitLog = ((Sys_Exhibit.DataTest) message.data()).getString();
        }
        else if(message.source() == Sys.ATS){
            atsLog = ((Sys_AutomatedTicket.DataTest) message.data()).getString();
        }
        else if(message.source() == Sys.OVERSIGHT){
            oversightLog = ((Sys_Oversight.DataTest) message.data()).getString();
        }
        else if(message.source() == Sys.AUTO_CAR){
            carLog = ((Sys_SelfDrivingCar.DataTest) message.data()).getString();
        }
    }

}

public class DataTest extends Data {
    private String text;

    public DataTest( String text) {
        this.text = text;
    }
    //getters and whatever
    public String getString(){
        return text;
    }
}
```

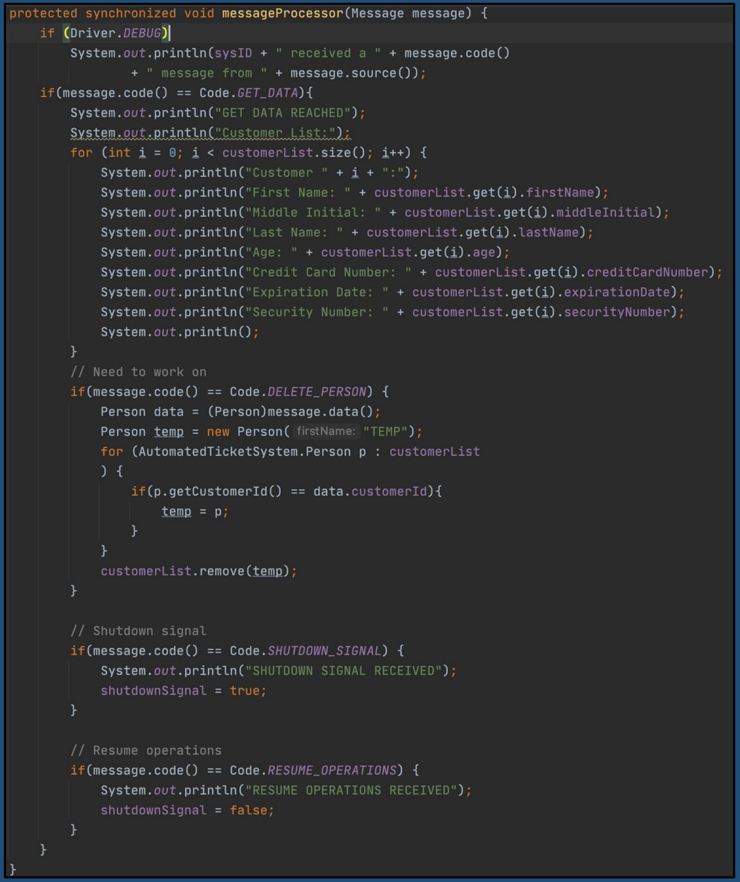


```java
Timer logTimer = new Timer();
logTimer.scheduleAtFixedRate(new TimerTask() {
    @Override
    public void run() {
        Platform.runLater(() ->{

            if(displayingLog){
                String newLogText = "";
                logArea.setMinHeight(400);
                logArea.setMaxWidth(600);
                logArea.setEditable(false);

                if(atsBtn.isSelected()){
                    newLogText = sgcController.getATSlog();
                }
                else if(carBtn.isSelected()){
                    newLogText = sgcController.getCarLog();
                }else if(exhibitBtn.isSelected()){
                    newLogText = sgcController.getExhibitLog();
                }else if(oversightBtn.isSelected()){
                    newLogText = sgcController.getOversightLog();
                }
                else if(carBtn.isSelected()){
                    newLogText = sgcController.getCarLog();
                }

                logArea.setText(newLogText);
                mainVBox.getChildren().clear();
                mainVBox.getChildren().addAll(btnnHBox, logArea);

                rootPane.setCenter(mainVBox);
            }

        });
    }
}, delay: 0, period: 2000);
```

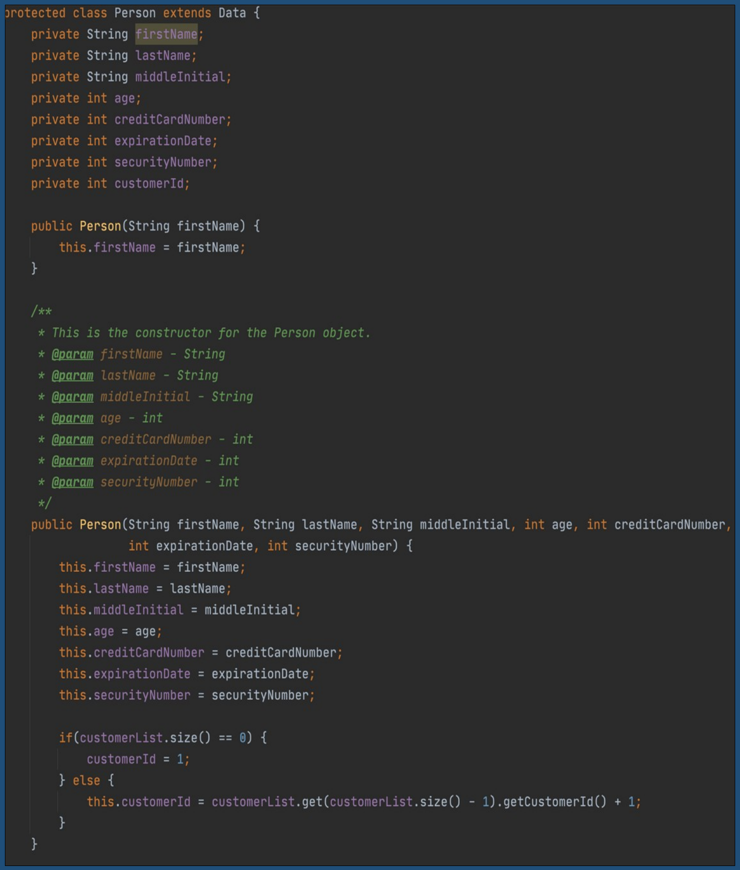
Components – Automated Ticket System (ATS)

- Contains Kiosk GUI that allows customers of the park to agree to waiver, submit payment information, and be verified for park access.
- Kiosk GUI sends information to ATS to verify payment.
- Once verified, a Person object is created to store customer information.
- ATS sends customer information to SGC.
- SGC can shutdown kiosks in case of emergency.

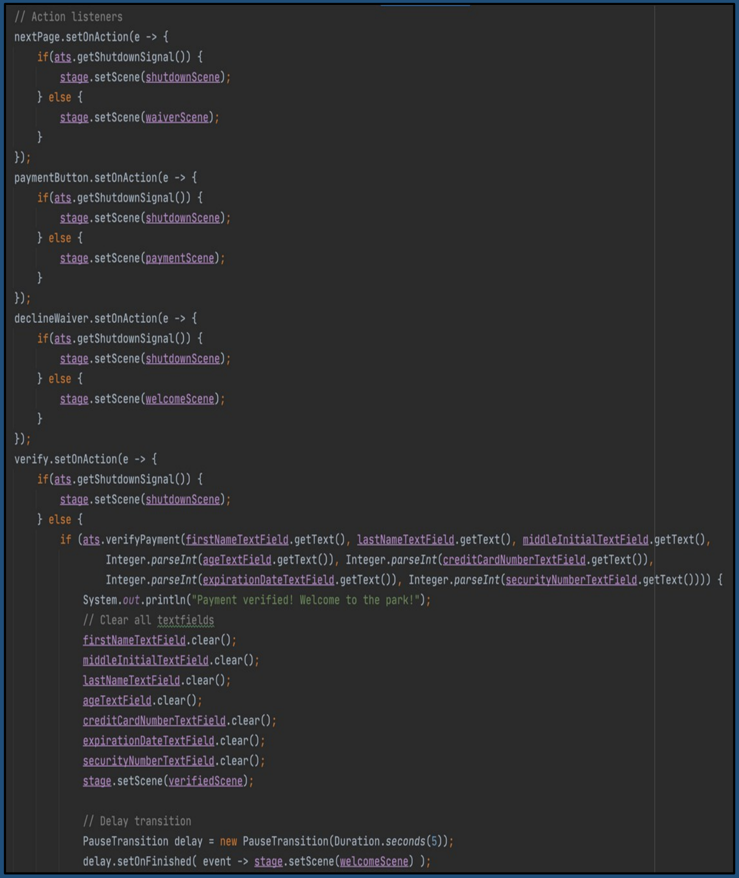
Automated Ticket System – Code Sample



```java
protected synchronized void messageProcessor(Message message) {
    if (Driver.DEBUG)
        System.out.println(sysID + " received a " + message.code()
                + " message from " + message.source());
    if(message.code() == Code.GET_DATA){
        System.out.println("GET DATA REACHED");
        System.out.println("Customer List:");
        for (int i = 0; i < customerList.size(); i++) {
            System.out.println("Customer " + i + ":");
            System.out.println("First Name: " + customerList.get(i).firstName);
            System.out.println("Middle Initial: " + customerList.get(i).middleInitial);
            System.out.println("Last Name: " + customerList.get(i).lastName);
            System.out.println("Age: " + customerList.get(i).age);
            System.out.println("Credit Card Number: " + customerList.get(i).creditCardNumber);
            System.out.println("Expiration Date: " + customerList.get(i).expirationDate);
            System.out.println("Security Number: " + customerList.get(i).securityNumber);
            System.out.println();
        }
        // Need to work on
        if(message.code() == Code.DELETE_PERSON) {
            Person data = (Person)message.data();
            Person temp = new Person( firstName: "TEMP");
            for (AutomatedTicketSystem.Person p : customerList
            ) {
                if(p.getCustomerId() == data.customerId){
                    temp = p;
                }
            }
            customerList.remove(temp);
        }

        // Shutdown signal
        if(message.code() == Code.SHUTDOWN_SIGNAL) {
            System.out.println("SHUTDOWN SIGNAL RECEIVED");
            shutdownSignal = true;
        }

        // Resume operations
        if(message.code() == Code.RESUME_OPERATIONS) {
            System.out.println("RESUME OPERATIONS RECEIVED");
            shutdownSignal = false;
        }
    }
}
```



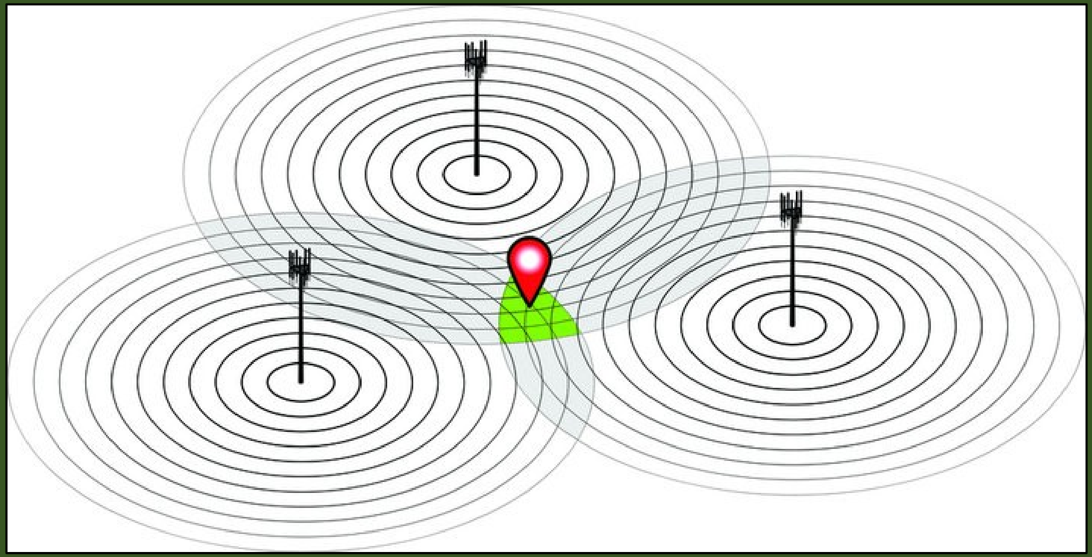
```java
protected class Person extends Data {
    private String firstName;
    private String lastName;
    private String middleInitial;
    private int age;
    private int creditCardNumber;
    private int expirationDate;
    private int securityNumber;
    private int customerId;

    public Person(String firstName) {
        this.firstName = firstName;
    }

    /**
     * This is the constructor for the Person object.
     * @param firstName - String
     * @param lastName - String
     * @param middleInitial - String
     * @param age - int
     * @param creditCardNumber - int
     * @param expirationDate - int
     * @param securityNumber - int
     */
    public Person(String firstName, String lastName, String middleInitial, int age, int creditCardNumber,
                  int expirationDate, int securityNumber) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.middleInitial = middleInitial;
        this.age = age;
        this.creditCardNumber = creditCardNumber;
        this.expirationDate = expirationDate;
        this.securityNumber = securityNumber;

        if(customerList.size() == 0) {
            customerId = 1;
        } else {
            this.customerId = customerList.get(customerList.size() - 1).getCustomerId() + 1;
        }
    }
```



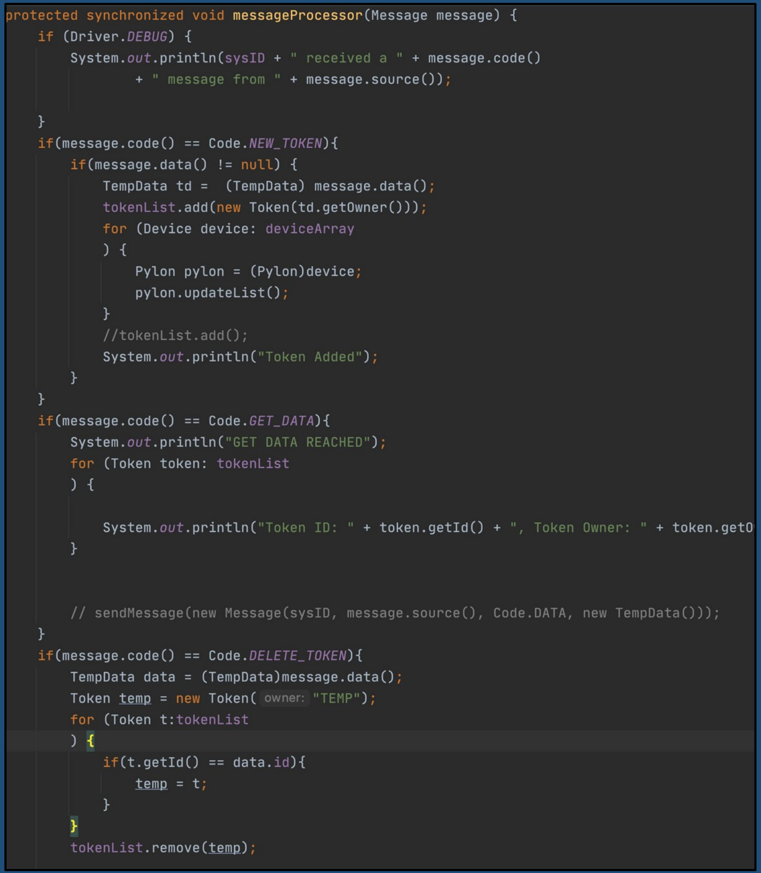
```java
// Action listeners
nextPage.setOnAction(e -> {
    if(ats.getShutdownSignal()) {
        stage.setScene(shutdownScene);
    } else {
        stage.setScene(waiverScene);
    }
});
paymentButton.setOnAction(e -> {
    if(ats.getShutdownSignal()) {
        stage.setScene(shutdownScene);
    } else {
        stage.setScene(paymentScene);
    }
});
declineWaiver.setOnAction(e -> {
    if(ats.getShutdownSignal()) {
        stage.setScene(shutdownScene);
    } else {
        stage.setScene(welcomeScene);
    }
});
verify.setOnAction(e -> {
    if(ats.getShutdownSignal()) {
        stage.setScene(shutdownScene);
    } else {
        if (ats.verifyPayment(firstNameTextField.getText(), lastNameTextField.getText(), middleInitialTextField.getText(),
                Integer.parseInt(ageTextField.getText()), Integer.parseInt(creditCardNumberTextField.getText()),
                Integer.parseInt(expirationDateTextField.getText()), Integer.parseInt(securityNumberTextField.getText()))) {
            System.out.println("Payment verified! Welcome to the park!");
            // Clear all textfields
            firstNameTextField.clear();
            middleInitialTextField.clear();
            lastNameTextField.clear();
            ageTextField.clear();
            creditCardNumberTextField.clear();
            expirationDateTextField.clear();
            securityNumberTextField.clear();
            stage.setScene(verifiedScene);

            // Delay transition
            PauseTransition delay = new PauseTransition(Duration.seconds(5));
            delay.setOnFinished( event -> stage.setScene(welcomeScene) );
```

Components – Token Monitoring System

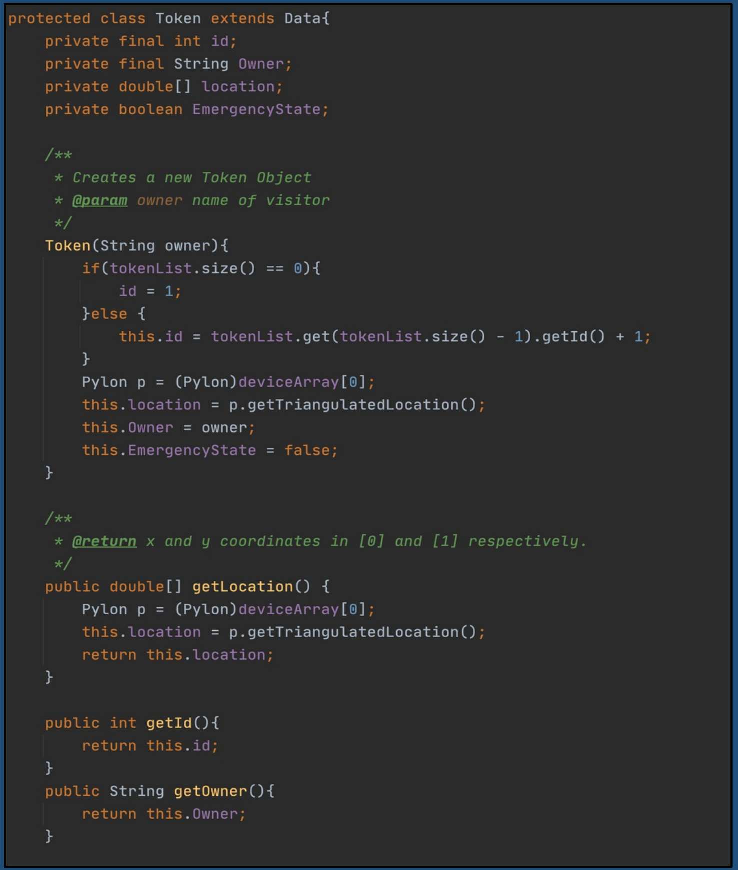
- Monitors the location of each active token in the park.
- Uses RFID Pylons throughout the park to track the Tokens given to each Visitor.
- Tokens are physical pieces of hardware with a RFID Chip inside them.
- Tokens are logged into the Token Monitoring System with its unique ID, Visitor Info, and its location.
- Location is updated as the RFID chips are read by the Pylons and then translated into X and Y coordinates.
- Maintains the generation and removal of new token IDs.



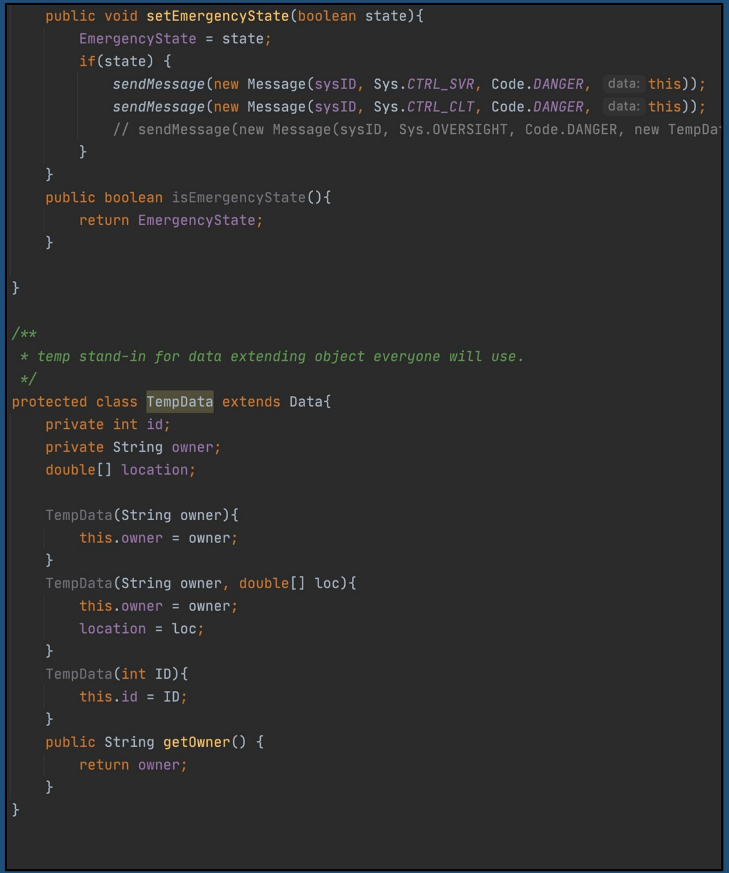
Token Monitoring System – Code Sample



```
protected synchronized void messageProcessor(Message message) {
    if (Driver.DEBUG) {
        System.out.println(sysID + " received a " + message.code()
                + " message from " + message.source());

    }
    if(message.code() == Code.NEW_TOKEN){
        if(message.data() != null) {
            TempData td =  (TempData) message.data();
            tokenList.add(new Token(td.getOwner()));
            for (Device device: deviceArray
            ) {
                Pylon pylon = (Pylon)device;
                pylon.updateList();
            }
            //tokenList.add();
            System.out.println("Token Added");
        }
    }
    if(message.code() == Code.GET_DATA){
        System.out.println("GET DATA REACHED");
        for (Token token: tokenList
        ) {

            System.out.println("Token ID: " + token.getId() + ", Token Owner: " + token.getO
        }



        // sendMessage(new Message(sysID, message.source(), Code.DATA, new TempData()));
    }
    if(message.code() == Code.DELETE_TOKEN){
        TempData data = (TempData)message.data();
        Token temp = new Token( owner: "TEMP");
        for (Token t:tokenList
        ) {
            if(t.getId() == data.id){
                temp = t;
            }
        }
        tokenList.remove(temp);
```



```
protected class Token extends Data{
    private final int id;
    private final String Owner;
    private double[] location;
    private boolean EmergencyState;

    /**
     * Creates a new Token Object
     * @param owner name of visitor
     */
    Token(String owner){
        if(tokenList.size() == 0){
            id = 1;
        }else {
            this.id = tokenList.get(tokenList.size() - 1).getId() + 1;
        }
        Pylon p = (Pylon)deviceArray[0];
        this.location = p.getTriangulatedLocation();
        this.Owner = owner;
        this.EmergencyState = false;
    }

    /**
     * @return x and y coordinates in [0] and [1] respectively.
     */
    public double[] getLocation() {
        Pylon p = (Pylon)deviceArray[0];
        this.location = p.getTriangulatedLocation();
        return this.location;
    }

    public int getId(){
        return this.id;
    }
    public String getOwner(){
        return this.Owner;
    }
```

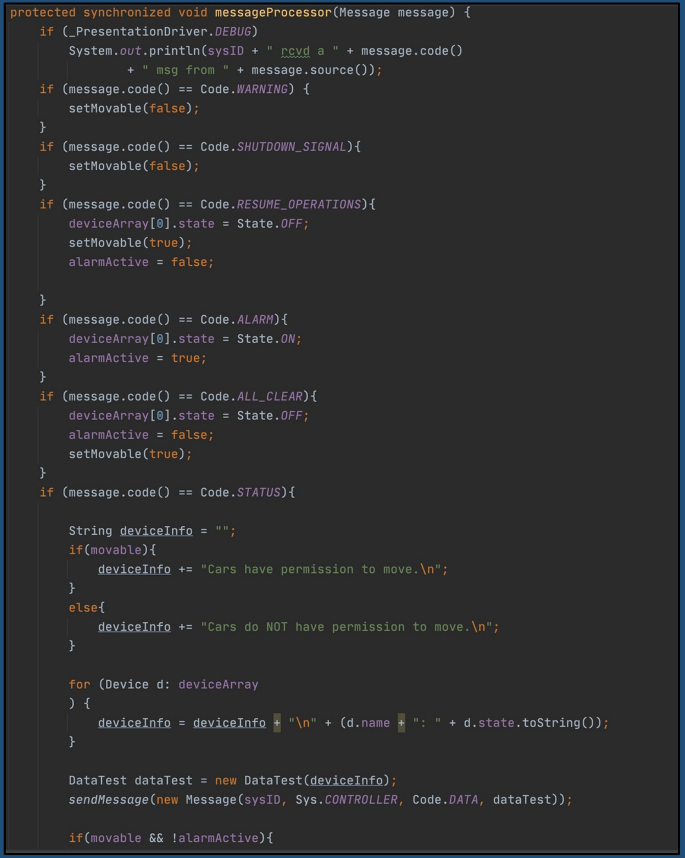


```
    public void setEmergencyState(boolean state){
        EmergencyState = state;
        if(state) {
            sendMessage(new Message(sysID, Sys.CTRL_SVR, Code.DANGER,  data: this));
            sendMessage(new Message(sysID, Sys.CTRL_CLT, Code.DANGER,  data: this));
            // sendMessage(new Message(sysID, Sys.OVERSIGHT, Code.DANGER, new TempDa
        }
    }
    public boolean isEmergencyState(){
        return EmergencyState;
    }

}

/**
 * temp stand-in for data extending object everyone will use.
 */
protected class TempData extends Data{
    private int id;
    private String owner;
    double[] location;

    TempData(String owner){
        this.owner = owner;
    }
    TempData(String owner, double[] loc){
        this.owner = owner;
        location = loc;
    }
    TempData(int ID){
        this.id = ID;
    }
    public String getOwner() {
        return owner;
    }
}
```

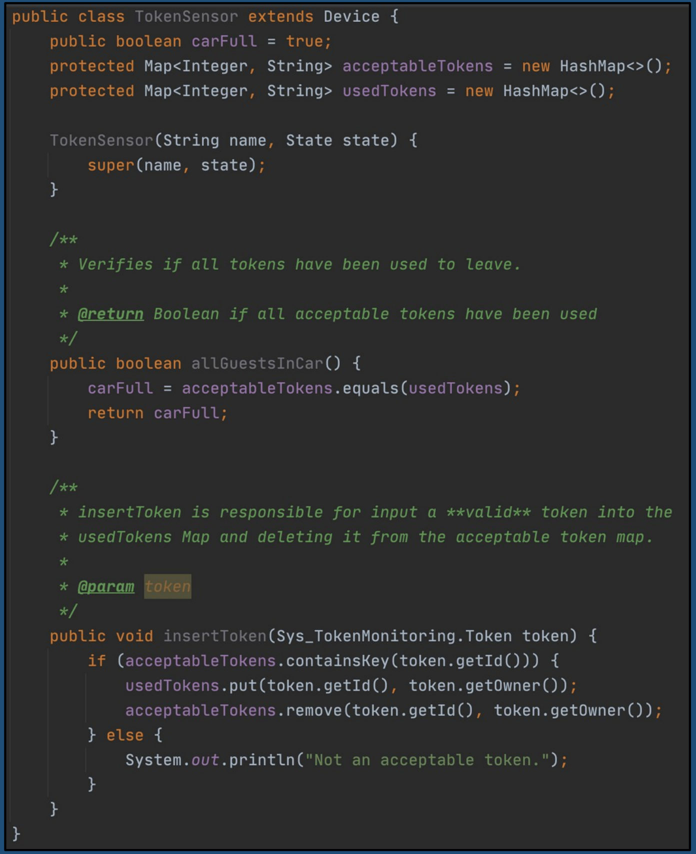
Components –Self Driving Car System

- System integrates data received from the self driving sensors on the cars.
- System checks to see if there are any obstructions and notifies the SGC through the message broker if there are any.
- Checks list of in use tokens and only allows those those passengers on.
- Manages all of the signals that can be sent from the SGC including shutdown signal that can be activated in case of emergency.

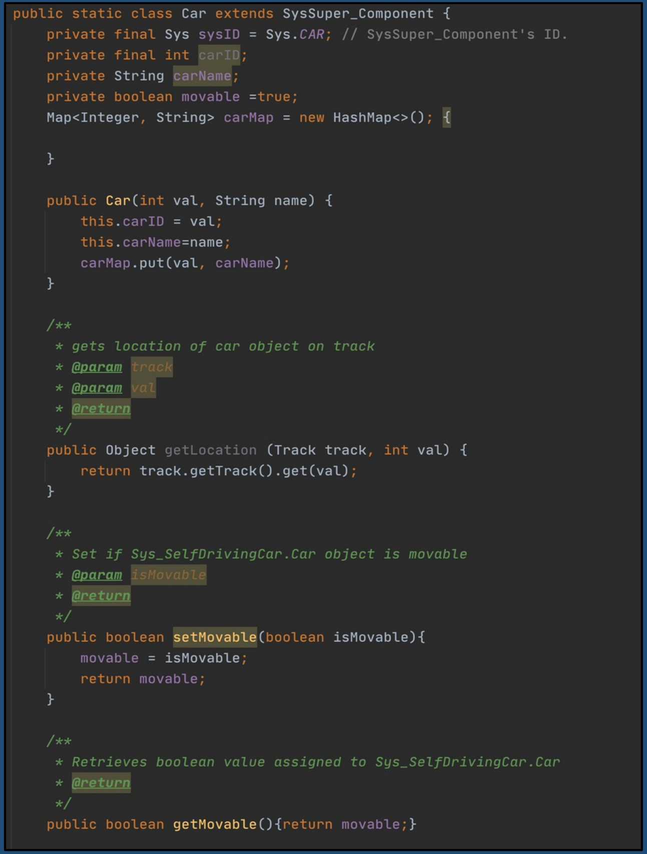
Self Driving Car System – Code Sample



```java
protected synchronized void messageProcessor(Message message) {
    if (_PresentationDriver.DEBUG)
        System.out.println(sysID + " rcvd a " + message.code()
                + " msg from " + message.source());
    if (message.code() == Code.WARNING) {
        setMovable(false);
    }
    if (message.code() == Code.SHUTDOWN_SIGNAL){
        setMovable(false);
    }
    if (message.code() == Code.RESUME_OPERATIONS){
        deviceArray[0].state = State.OFF;
        setMovable(true);
        alarmActive = false;

    }
    if (message.code() == Code.ALARM){
        deviceArray[0].state = State.ON;
        alarmActive = true;
    }
    if (message.code() == Code.ALL_CLEAR){
        deviceArray[0].state = State.OFF;
        alarmActive = false;
        setMovable(true);
    }
    if (message.code() == Code.STATUS){

        String deviceInfo = "";
        if(movable){
            deviceInfo += "Cars have permission to move.\n";
        }
        else{
            deviceInfo += "Cars do NOT have permission to move.\n";
        }

        for (Device d: deviceArray
        ) {
            deviceInfo = deviceInfo + "\n" + (d.name + ": " + d.state.toString());
        }

        DataTest dataTest = new DataTest(deviceInfo);
        sendMessage(new Message(sysID, Sys.CONTROLLER, Code.DATA, dataTest));

        if(movable && !alarmActive){
```



```java
public class TokenSensor extends Device {
    public boolean carFull = true;
    protected Map<Integer, String> acceptableTokens = new HashMap<>();
    protected Map<Integer, String> usedTokens = new HashMap<>();

    TokenSensor(String name, State state) {
        super(name, state);
    }

    /**
     * Verifies if all tokens have been used to leave.
     *
     * @return Boolean if all acceptable tokens have been used
     */
    public boolean allGuestsInCar() {
        carFull = acceptableTokens.equals(usedTokens);
        return carFull;
    }

    /**
     * insertToken is responsible for input a **valid** token into the
     * usedTokens Map and deleting it from the acceptable token map.
     *
     * @param token
     */
    public void insertToken(Sys_TokenMonitoring.Token token) {
        if (acceptableTokens.containsKey(token.getId())) {
            usedTokens.put(token.getId(), token.getOwner());
            acceptableTokens.remove(token.getId(), token.getOwner());
        } else {
            System.out.println("Not an acceptable token.");
        }
    }
}
```

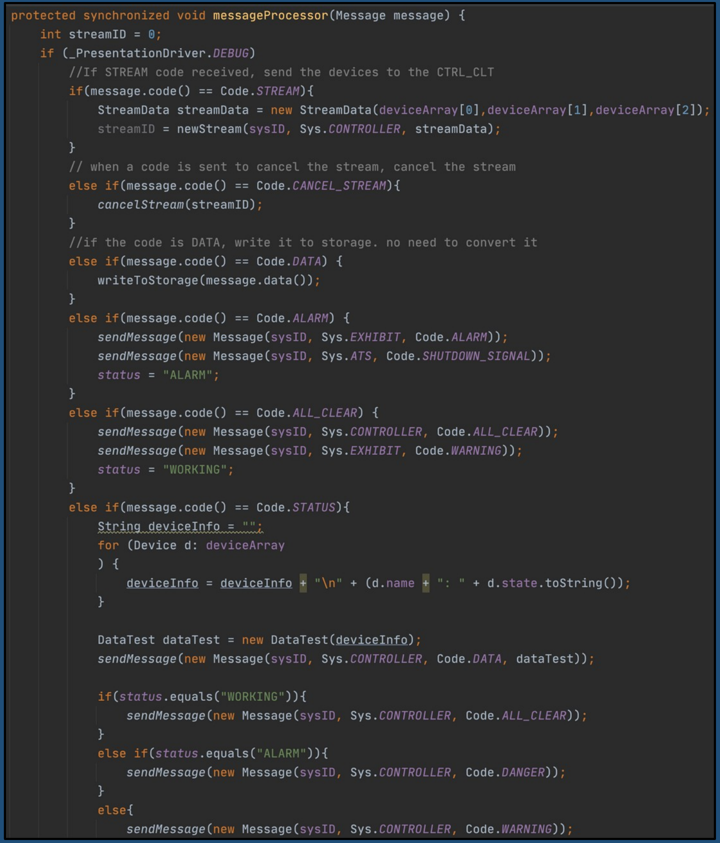


```java
public static class Car extends SysSuper_Component {
    private final Sys sysID = Sys.CAR; // SysSuper_Component's ID.
    private final int carID;
    private String carName;
    private boolean movable =true;
    Map<Integer, String> carMap = new HashMap<>(); {

    }

    public Car(int val, String name) {
        this.carID = val;
        this.carName=name;
        carMap.put(val, carName);
    }

    /**
     * gets location of car object on track
     * @param track
     * @param val
     * @return
     */
    public Object getLocation (Track track, int val) {
        return track.getTrack().get(val);
    }

    /**
     * Set if Sys_SelfDrivingCar.Car object is movable
     * @param isMovable
     * @return
     */
    public boolean setMovable(boolean isMovable){
        movable = isMovable;
        return movable;
    }

    /**
     * Retrieves boolean value assigned to Sys_SelfDrivingCar.Car
     * @return
     */
    public boolean getMovable(){return movable;}
```

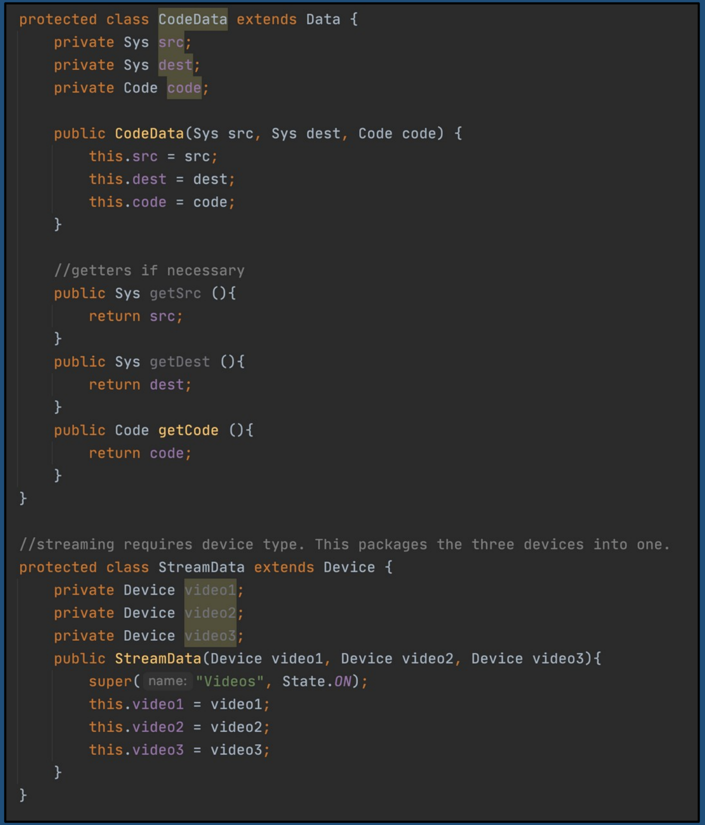
Components – Oversight System

- Serves as a means of communication with the SGC and as a way to directly monitor video feeds and alarm systems.
- GUI allows guard stations throughout the park to monitor the live feed from various cameras around the park and the general status of the alarms.
- Camera feeds use a Stream to provide constant data to the SGC.
- Defines the reactions to commands coming in from the message broker.

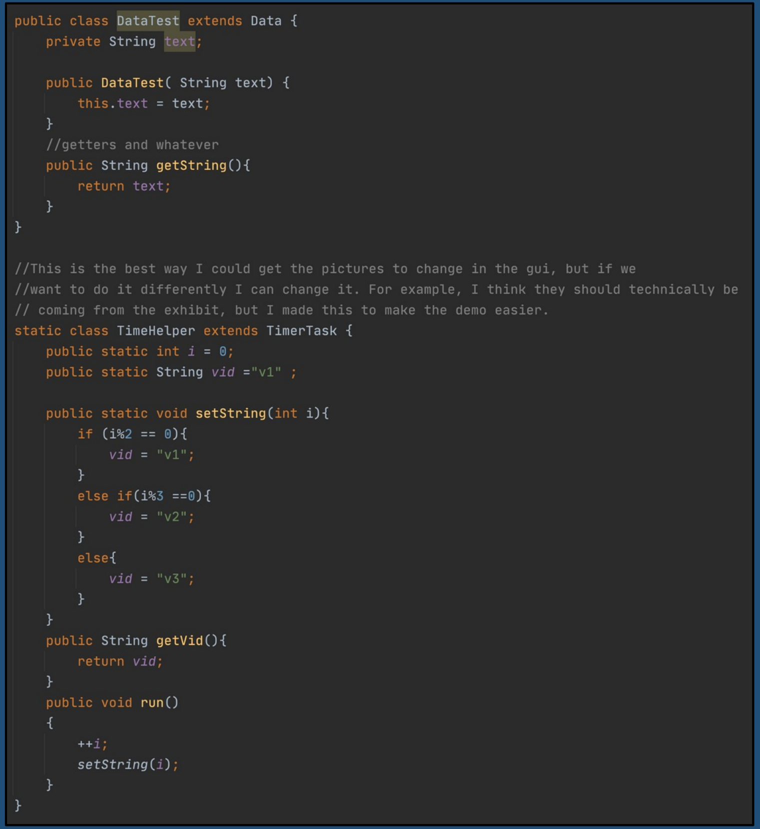
Oversight System – Code Sample



```java
protected synchronized void messageProcessor(Message message) {
    int streamID = 0;
    if (_PresentationDriver.DEBUG)
        //If STREAM code received, send the devices to the CTRL_CLT
        if(message.code() == Code.STREAM){
            StreamData streamData = new StreamData(deviceArray[0],deviceArray[1],deviceArray[2]);
            streamID = newStream(sysID, Sys.CONTROLLER, streamData);
        }
        // when a code is sent to cancel the stream, cancel the stream
        else if(message.code() == Code.CANCEL_STREAM){
            cancelStream(streamID);
        }
        //if the code is DATA, write it to storage. no need to convert it
        else if(message.code() == Code.DATA) {
            writeToStorage(message.data());
        }
        else if(message.code() == Code.ALARM) {
            sendMessage(new Message(sysID, Sys.EXHIBIT, Code.ALARM));
            sendMessage(new Message(sysID, Sys.ATS, Code.SHUTDOWN_SIGNAL));
            status = "ALARM";
        }
        else if(message.code() == Code.ALL_CLEAR) {
            sendMessage(new Message(sysID, Sys.CONTROLLER, Code.ALL_CLEAR));
            sendMessage(new Message(sysID, Sys.EXHIBIT, Code.WARNING));
            status = "WORKING";
        }
        else if(message.code() == Code.STATUS){
            String deviceInfo = "";
            for (Device d: deviceArray
            ) {
                deviceInfo = deviceInfo + "\n" + (d.name + ": " + d.state.toString());
            }

            DataTest dataTest = new DataTest(deviceInfo);
            sendMessage(new Message(sysID, Sys.CONTROLLER, Code.DATA, dataTest));

            if(status.equals("WORKING")){
                sendMessage(new Message(sysID, Sys.CONTROLLER, Code.ALL_CLEAR));
            }
            else if(status.equals("ALARM")){
                sendMessage(new Message(sysID, Sys.CONTROLLER, Code.DANGER));
            }
            else{
                sendMessage(new Message(sysID, Sys.CONTROLLER, Code.WARNING));
```



```java
protected class CodeData extends Data {
    private Sys src;
    private Sys dest;
    private Code code;

    public CodeData(Sys src, Sys dest, Code code) {
        this.src = src;
        this.dest = dest;
        this.code = code;
    }

    //getters if necessary
    public Sys getSrc (){
        return src;
    }
    public Sys getDest (){
        return dest;
    }
    public Code getCode (){
        return code;
    }
}

//streaming requires device type. This packages the three devices into one.
protected class StreamData extends Device {
    private Device video1;
    private Device video2;
    private Device video3;
    public StreamData(Device video1, Device video2, Device video3){
        super( name: "Videos", State.ON);
        this.video1 = video1;
        this.video2 = video2;
        this.video3 = video3;
    }
}
```

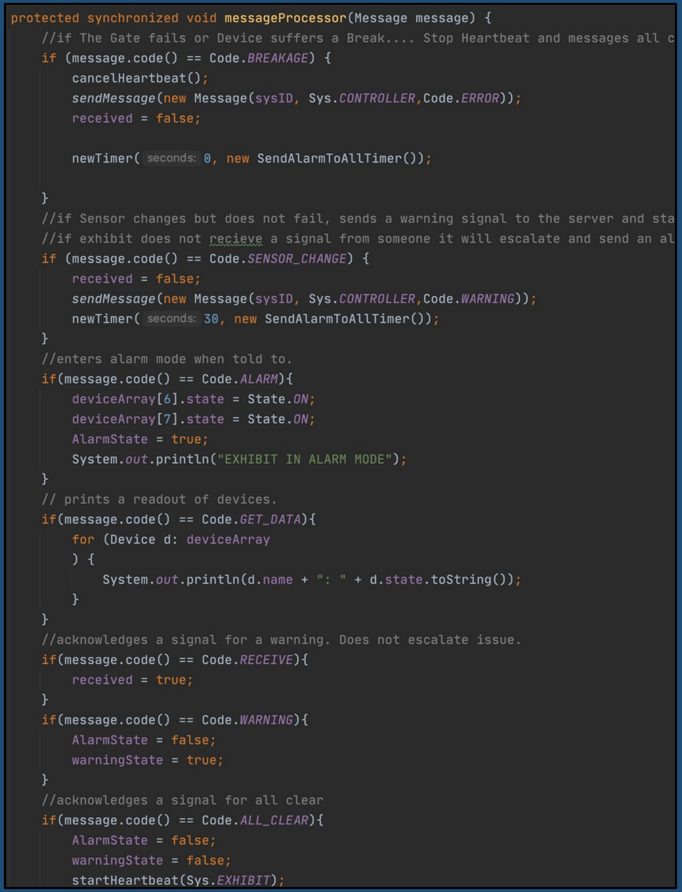


```java
public class DataTest extends Data {
    private String text;

    public DataTest( String text) {
        this.text = text;
    }
    //getters and whatever
    public String getString(){
        return text;
    }
}

//This is the best way I could get the pictures to change in the gui, but if we
//want to do it differently I can change it. For example, I think they should technically be
// coming from the exhibit, but I made this to make the demo easier.
static class TimeHelper extends TimerTask {
    public static int i = 0;
    public static String vid ="v1" ;

    public static void setString(int i){
        if (i%2 == 0){
            vid = "v1";
        }
        else if(i%3 ==0){
            vid = "v2";
        }
        else{
            vid = "v3";
        }
    }
    public String getVid(){
        return vid;
    }
    public void run()
    {
        ++i;
        setString(i);
    }
}
```

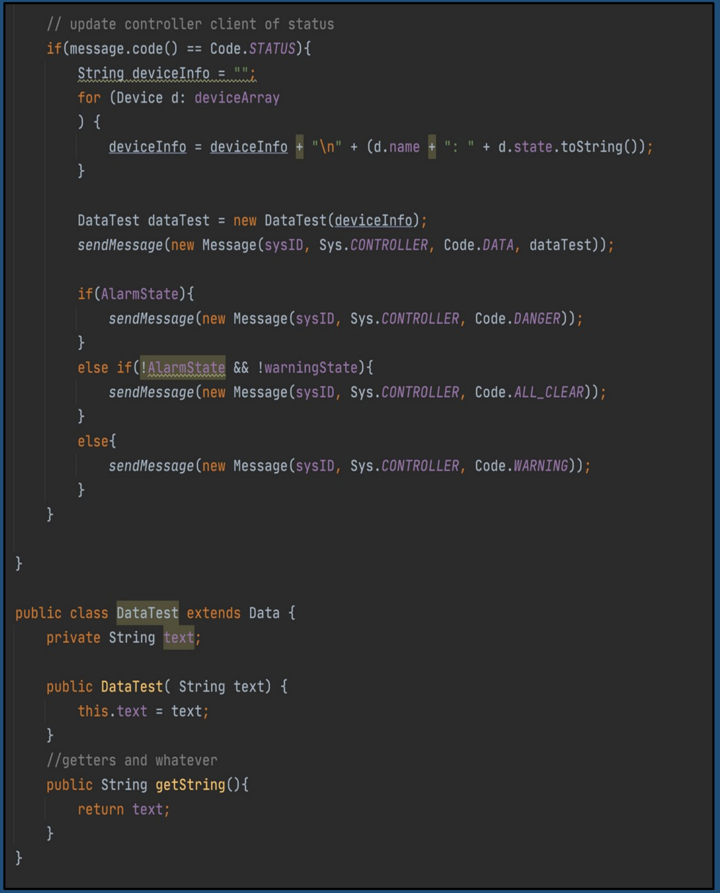
Components – Exhibit System

- The Exhibit Component is a subclass of the Component superclass.
- The Exhibit communicates the status of the T-Rex Enclosure and its devices through the use of heartbeat signals.
- In the event of a catastrophic failure to the exhibit, an Alarm will be sent out to all Components for the safety of Siesta Garden Visitors.

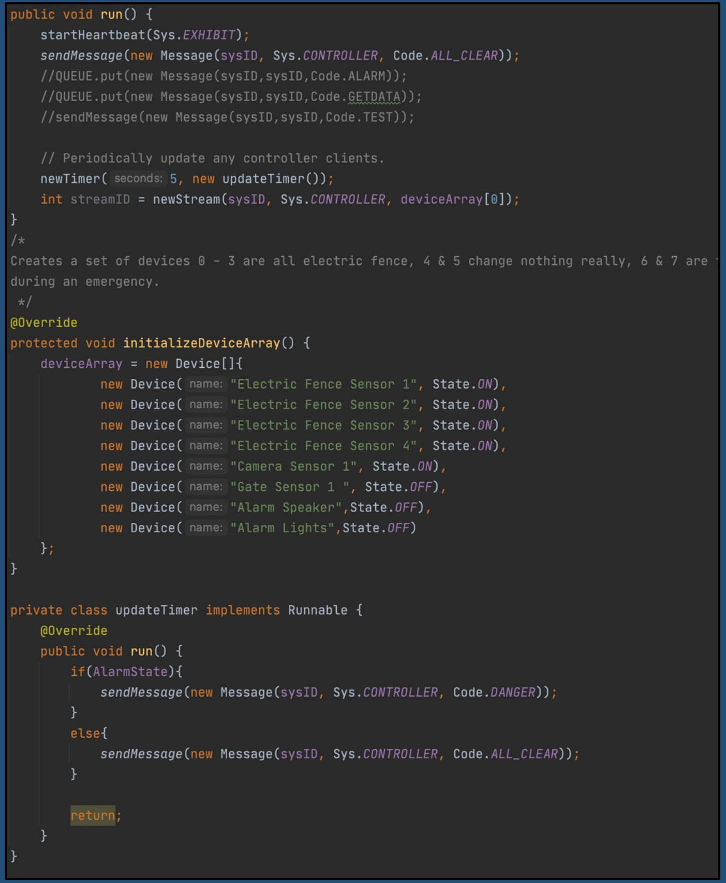
Exhibit System – Code Sample



```java
protected synchronized void messageProcessor(Message message) {
    //if The Gate fails or Device suffers a Break.... Stop Heartbeat and messages all c
    if (message.code() == Code.BREAKAGE) {
        cancelHeartbeat();
        sendMessage(new Message(sysID, Sys.CONTROLLER,Code.ERROR));
        received = false;

        newTimer( seconds: 0, new SendAlarmToAllTimer());

    }
    //if Sensor changes but does not fail, sends a warning signal to the server and sta
    //if exhibit does not recieve a signal from someone it will escalate and send an al
    if (message.code() == Code.SENSOR_CHANGE) {
        received = false;
        sendMessage(new Message(sysID, Sys.CONTROLLER,Code.WARNING));
        newTimer( seconds: 30, new SendAlarmToAllTimer());
    }
    //enters alarm mode when told to.
    if(message.code() == Code.ALARM){
        deviceArray[6].state = State.ON;
        deviceArray[7].state = State.ON;
        AlarmState = true;
        System.out.println("EXHIBIT IN ALARM MODE");
    }
    // prints a readout of devices.
    if(message.code() == Code.GET_DATA){
        for (Device d: deviceArray
        ) {
            System.out.println(d.name + ": " + d.state.toString());
        }
    }
    //acknowledges a signal for a warning. Does not escalate issue.
    if(message.code() == Code.RECEIVE){
        received = true;
    }
    if(message.code() == Code.WARNING){
        AlarmState = false;
        warningState = true;
    }
    //acknowledges a signal for all clear
    if(message.code() == Code.ALL_CLEAR){
        AlarmState = false;
        warningState = false;
        startHeartbeat(Sys.EXHIBIT);
```



```java
    // update controller client of status
    if(message.code() == Code.STATUS){
        String deviceInfo = "";
        for (Device d: deviceArray
        ) {
            deviceInfo = deviceInfo + "\n" + (d.name + ": " + d.state.toString());
        }

        DataTest dataTest = new DataTest(deviceInfo);
        sendMessage(new Message(sysID, Sys.CONTROLLER, Code.DATA, dataTest));

        if(AlarmState){
            sendMessage(new Message(sysID, Sys.CONTROLLER, Code.DANGER));
        }
        else if(!AlarmState && !warningState){
            sendMessage(new Message(sysID, Sys.CONTROLLER, Code.ALL_CLEAR));
        }
        else{
            sendMessage(new Message(sysID, Sys.CONTROLLER, Code.WARNING));
        }
    }

}

public class DataTest extends Data {
    private String text;

    public DataTest( String text) {
        this.text = text;
    }
    //getters and whatever
    public String getString(){
        return text;
    }
}
```



```java
public void run() {
    startHeartbeat(Sys.EXHIBIT);
    sendMessage(new Message(sysID, Sys.CONTROLLER, Code.ALL_CLEAR));
    //QUEUE.put(new Message(sysID,sysID,Code.ALARM));
    //QUEUE.put(new Message(sysID,sysID,Code.GETDATA));
    //sendMessage(new Message(sysID,sysID,Code.TEST));

    // Periodically update any controller clients.
    newTimer( seconds: 5, new updateTimer());
    int streamID = newStream(sysID, Sys.CONTROLLER, deviceArray[0]);
}
/*
Creates a set of devices 0 - 3 are all electric fence, 4 & 5 change nothing really, 6 & 7 are
during an emergency.
 */
@Override
protected void initializeDeviceArray() {
    deviceArray = new Device[]{
            new Device( name: "Electric Fence Sensor 1", State.ON),
            new Device( name: "Electric Fence Sensor 2", State.ON),
            new Device( name: "Electric Fence Sensor 3", State.ON),
            new Device( name: "Electric Fence Sensor 4", State.ON),
            new Device( name: "Camera Sensor 1", State.ON),
            new Device( name: "Gate Sensor 1 ", State.OFF),
            new Device( name: "Alarm Speaker",State.OFF),
            new Device( name: "Alarm Lights",State.OFF)
    };
}

private class updateTimer implements Runnable {
    @Override
    public void run() {
        if(AlarmState){
            sendMessage(new Message(sysID, Sys.CONTROLLER, Code.DANGER));
        }
        else{
            sendMessage(new Message(sysID, Sys.CONTROLLER, Code.ALL_CLEAR));
        }

        return;
    }
}
```

Demonstration