# **Programming Assignment Unit 5**

Computer Science, University of the People

CS 1101-01 Programming Fundamentals - CS 1102-01 - AY2024-T2

Instructor, Noman Shihadeh

December 27, 2023

## Vehicle Rental Management System

For this assignment, we were asked to write a program that manages a collection of vehicles for a rental company. We need to build the solution using OOP and encapsulation principles. The system will have the following controls and abilities:

1. Vehicle types:
   1. **Shuttle** – “Car”.
   2. **Starfighter** – “Motorcycle”.
   3. **Cargo Ship** – “Truck”.
2. Management Controls:
   1. **Add** – Allows adding a vehicle to the collection.
   2. **Edit** – Allows selecting and editing a vehicle's details.
   3. **Delete** – Removes a selected vehicle from the collection.
   4. **List** – List all the vehicles in the collection with all their details.
   5. **List Simple** – List only the name and type of all the vehicles in the collection.
   6. **Exit** – Exits the application.

Source Code:

// MainSystem.java

package unit6;

import unit6.models.Starfighter;

import unit6.models.Shuttle;

import unit6.interfaces.Spacecraft;

import unit6.models.CargoShip;

import unit6.utilities.FormatUtils;

import unit6.utilities.InputValidators;

import unit6.utilities.SampleData;

import java.util.ArrayList;

import java.util.Scanner;

/\*\*

 \* Main class for the program to manage spacecraft rentals

 \*/

public class MainSystem {

    private static final Scanner scanner = new Scanner(System.in);

    private static final ArrayList<Spacecraft> spacecrafts = new ArrayList<>();

    /\*\*

     \* Main entry function for the program

     \*

     \* @param args the command line arguments

     \*/

    public static void main(String[] args) {

        System.err.println(FormatUtils.hr());

        System.out.println(FormatUtils.centerString("Intergalactic Voyages: Simon's Starship Bazaar"));

        System.out.println(FormatUtils.centerString("Zooming Through the Cosmos at Warp Speed!"));

        System.out.println(FormatUtils.centerString("Where Space Oddities Become Your Reality"));

        System.out.println(FormatUtils.centerString("\* All sales are final. No refunds or exchanges \*"));

        System.err.println(FormatUtils.hr());

        // Load sample data

        SampleData.getAllSamples(spacecrafts);

        boolean running = true;

        while (running) {

            displayMenu();

            String choice = scanner.nextLine().toUpperCase();

            switch (choice) {

                case "1":

                    addSpacecraft();

                    break;

                case "2":

                    editSpacecraft();

                    break;

                case "3":

                    deleteSpacecraft();

                    break;

                case "4":

                    listAllSpacecrafts();

                    break;

                case "5":

                    listAllSpacecraftsSimple();

                    break;

                case "E":

                    running = false;

                    break;

                default:

                    System.out.println("Invalid option. Please try again.");

            }

        }

        System.out.println("Thank you for using Simon's Starship Bazaar!");

        System.out.println("Goodbye. Live long and prosper. NO REFUNDS.");

    }

    /\*\*

     \* Method to list all spacecrafts (simple)

     \*/

    private static void listAllSpacecraftsSimple() {

        System.out.println(FormatUtils.hr("=", " List of All Spacecrafts (Simple) "));

        StringBuilder list = new StringBuilder();

        for (Spacecraft spacecraft : spacecrafts) {

            list.append(spacecraft.getTypeName() + " - " + spacecraft.getName() + "\n");

        }

        System.out.println(list.toString());

    }

    /\*\*

     \* Method to select a spacecraft from the list

     \*

     \* @param actionTitle the title of the action to perform

     \* @return the selected spacecraft

     \*/

    private static Spacecraft selectSpacecraft(String actionTitle) {

        System.out.println();

        System.out.println(FormatUtils.hr("=", " " + actionTitle + " "));

        if (spacecrafts.isEmpty()) {

            System.out.println("No spacecrafts available.");

            return null;

        }

        System.out.println("Select a spacecraft to " + actionTitle.toLowerCase() + ":");

        for (int i = 0; i < spacecrafts.size(); i++) {

            Spacecraft spacecraft = spacecrafts.get(i);

            System.out.println((i + 1) + ". " + spacecraft.getName() + " (" + spacecraft.getTypeName() + ")");

        }

        System.out.println("B. Back to Main Menu");

        System.out.print("Enter your choice: ");

        String choice = scanner.nextLine().toUpperCase();

        if (choice.equals("B")) {

            return null;

        }

        int index;

        try {

            index = Integer.parseInt(choice) - 1;

        } catch (NumberFormatException e) {

            System.out.println("Invalid option. Please try again.");

            return null;

        }

        if (index < 0 || index >= spacecrafts.size()) {

            System.out.println("Invalid option. Please try again.");

            return null;

        }

        return spacecrafts.get(index);

    }

    /\*\*

     \* Method to delete a spacecraft

     \*/

    private static void deleteSpacecraft() {

        Spacecraft spacecraft = selectSpacecraft("Delete");

        if (spacecraft != null) {

            String type = spacecraft.getTypeName();

            System.out.println("Are you sure you want to delete the " + type + " '"

                    + spacecraft.getName() + "'? (yes/no)");

            if (InputValidators.getYesNoInput(scanner, "")) {

                spacecrafts.remove(spacecraft);

                System.out.println(type + " deleted successfully!");

            } else {

                System.out.println(type + " deletion aborted.");

            }

        }

    }

    /\*\*

     \* Method to edit a spacecraft

     \*/

    private static void editSpacecraft() {

        Spacecraft spacecraft = selectSpacecraft("Edit");

        if (spacecraft != null) {

            spacecraft.editFromInput(scanner);

        }

    }

    /\*\*

     \* Method to display the main menu

     \*/

    private static void displayMenu() {

        System.out.println();

        System.out.println(FormatUtils.hr("=", " Main Menu "));

        System.out.println("1. Add Spacecraft");

        System.out.println("2. Edit Spacecraft");

        System.out.println("3. Delete Spacecraft");

        System.out.println("4. List All Spacecrafts (Detailed)");

        System.out.println("5. List All Spacecrafts (Simple)");

        System.out.println("E. Exit");

        System.out.print("Enter your choice: ");

    }

    /\*\*

     \* Method to add a spacecraft

     \*/

    private static void addSpacecraft() {

        System.out.println();

        System.out.println(FormatUtils.hr("=", " Select type to add "));

        System.out.println("1. Starfighter");

        System.out.println("2. Shuttle");

        System.out.println("3. Cargo Ship");

        System.out.println("B. Back to Main Menu");

        System.out.print("Enter your choice: ");

        String choice = scanner.nextLine().toUpperCase();

        Spacecraft spacecraft = null;

        switch (choice) {

            case "1":

                spacecraft = Starfighter.createStarfighterFromInput(scanner);

                break;

            case "2":

                spacecraft = Shuttle.createShuttleFromInput(scanner);

                break;

            case "3":

                spacecraft = CargoShip.createCargoShipFromInput(scanner);

                break;

            case "E":

                return;

            default:

                System.out.println("Invalid option. Please try again.");

        }

        if (spacecraft != null) {

            spacecrafts.add(spacecraft);

            System.out.println("Spacecraft added successfully!");

        }

    }

    /\*\*

     \* Method to list all spacecrafts

     \*/

    private static void listAllSpacecrafts() {

        System.out.println(FormatUtils.hr("=", " List of All Spacecrafts "));

        System.out.println();

        boolean hasStarfighters = false, hasShuttles = false, hasCargoShips = false;

        StringBuilder starfightersList = new StringBuilder();

        StringBuilder shuttlesList = new StringBuilder();

        StringBuilder cargoShipsList = new StringBuilder();

        for (Spacecraft spacecraft : spacecrafts) {

            if (spacecraft instanceof Starfighter) {

                hasStarfighters = true;

                starfightersList.append(spacecraft.toString());

            } else if (spacecraft instanceof Shuttle) {

                hasShuttles = true;

                shuttlesList.append(spacecraft.toString());

            } else if (spacecraft instanceof CargoShip) {

                hasCargoShips = true;

                cargoShipsList.append(spacecraft.toString());

            }

        }

        System.out.println(FormatUtils.hr("=", " Starfighters "));

        System.out.println();

        System.out.println(hasStarfighters ? starfightersList.toString() : "Out of Stock");

        System.out.println(FormatUtils.hr("=", " Shuttles "));

        System.out.println();

        System.out.println(hasShuttles ? shuttlesList.toString() : "Out of Stock");

        System.out.println(FormatUtils.hr("=", " Cargo Ships "));

        System.out.println();

        System.out.println(hasCargoShips ? cargoShipsList.toString() : "Out of Stock");

    }

}

// ---------------------------------------------------------------------------

// interfaces/Spacecraft.java

package unit6.interfaces;

import java.util.Scanner;

/\*\*

 \* Interface for a spacecraft

 \*/

public interface Spacecraft {

    /\*\*

     \* Method to get the name of the spacecraft

     \*

     \* @return the name of the spacecraft

     \*/

    String getName();

    /\*\*

     \* Method to set the name of the spacecraft

     \*

     \* @param name the name of the spacecraft

     \*/

    void setName(String name);

    /\*\*

     \* Method to get the model of the spacecraft

     \*

     \* @return the model of the spacecraft

     \*/

    String getModel();

    /\*\*

     \* Method to set the model of the spacecraft

     \*

     \* @param model the model of the spacecraft

     \*/

    void setModel(String model);

    /\*\*

     \* Method to get the type of engine used by the spacecraft

     \*

     \* @return the type of engine used by the spacecraft

     \*/

    String getEngineType();

    /\*\*

     \* Method to set the type of engine used by the spacecraft

     \*

     \* @param engineType the type of engine used by the spacecraft

     \*/

    void setEngineType(String engineType);

    /\*\*

     \* Method to get the rentable status of the spacecraft

     \*

     \* @return the rentable status of the spacecraft

     \*/

    boolean getIsRentable();

    /\*\*

     \* Method to set the rentable status of the spacecraft

     \*

     \* @return the rentable status of the spacecraft

     \*/

    double getRentalCostPerParsec();

    /\*\*

     \* Method to set the rentable status of the spacecraft

     \*

     \* @param isRentable the rentable status of the spacecraft

     \*/

    void setIsRentable(boolean isRentable);

    /\*\*

     \* Method to get the rental cost per parsec of the spacecraft

     \*

     \* @param rentalCostPerParsec the rental cost per parsec of the spacecraft

     \*/

    void setRentalCostPerParsec(double rentalCostPerParsec);

    /\*\*

     \* Method to get the weapon system of the spacecraft

     \*

     \* @param scanner the scanner to use for user input

     \*/

    void editFromInput(Scanner scanner);

    /\*\*

     \* Method to get the type name of the spacecraft

     \*

     \* @return the type name of the spacecraft

     \*/

    String getTypeName();

}

// ---------------------------------------------------------------------------

// interfaces/Spacecraft.java

package unit6.interfaces;

/\*\*

 \* Interface for a shuttle spacecraft

 \*/

public interface ShuttleSpacecraft extends Spacecraft {

    /\*\*

     \* Method to get the crew capacity of the shuttle

     \*

     \* @return the crew capacity of the shuttle

     \*/

    int getCrewCapacity();

    /\*\*

     \* Method to set the crew capacity of the shuttle

     \*

     \* @param crewCapacity the crew capacity of the shuttle

     \*/

    void setCrewCapacity(int crewCapacity);

    /\*\*

     \* Method to get the mission type for the shuttle

     \*

     \* @return the mission type for the shuttle

     \*/

    String getMissionType();

    /\*\*

     \* Method to set the mission type for the shuttle

     \*

     \* @param missionType the mission type for the shuttle

     \*/

    void setMissionType(String missionType);

}

// ---------------------------------------------------------------------------

// interfaces/StarfighterSpacecraft.java

package unit6.interfaces;

/\*\*

 \* Interface for a starfighter spacecraft

 \*/

public interface StarfighterSpacecraft extends Spacecraft {

    /\*\*

     \* Method to get the weapon system of the starfighter

     \*

     \* @return the weapon system of the starfighter

     \*/

    String getWeaponSystem();

    /\*\*

     \* Method to set the weapon system of the starfighter

     \*

     \* @param weaponSystem the weapon system of the starfighter

     \*/

    void setWeaponSystem(String weaponSystem);

    /\*\*

     \* Method to get the hyperdrive capability of the starfighter

     \*

     \* @return the hyperdrive capability of the starfighter

     \*/

    boolean hasHyperdriveCapability();

    /\*\*

     \* Method to set the hyperdrive capability of the starfighter

     \*

     \* @param hyperdriveCapability the hyperdrive capability of the starfighter

     \*/

    void setHyperdriveCapability(boolean hyperdriveCapability);

}

// ---------------------------------------------------------------------------

// interfaces/ShuttleSpacecraft.java

package unit6.interfaces;

/\*\*

 \* Interface for a cargo ship spacecraft

 \*/

public interface CargoShipSpacecraft extends Spacecraft {

    /\*\*

     \* Method to get the cargo capacity of the cargo ship

     \*

     \* @return the cargo capacity of the cargo ship

     \*/

    double getCargoCapacity();

    /\*\*

     \* Method to set the cargo capacity of the cargo ship

     \*

     \* @param cargoCapacity the cargo capacity of the cargo ship

     \*/

    void setCargoCapacity(double cargoCapacity);

    /\*\*

     \* Method to get the presence of a specialized cargo handling system in the

     \* cargo ship

     \*

     \* @return the presence of a specialized cargo handling system in the cargo ship

     \*/

    boolean hasSpecializedCargoHandling();

    /\*\*

     \* Method to set the presence of a specialized cargo handling system in the

     \* cargo ship

     \*

     \* @param specializedCargoHandling the presence of a specialized cargo handling

     \*                                 system in the cargo ship

     \*/

    void setSpecializedCargoHandling(boolean specializedCargoHandling);

}

// ---------------------------------------------------------------------------

// models/BaseSpacecraft.java

package unit6.models;

import java.util.Scanner;

import unit6.utilities.FormatUtils;

import unit6.utilities.InputValidators;

/\*\*

 \* Abstract class for a spacecraft

 \*/

public abstract class BaseSpacecraft {

    protected String name;

    protected String model;

    protected String engineType;

    protected boolean isRentable;

    protected double rentalCostPerParsec;

    /\*\*

     \* Default constructor

     \*/

    public BaseSpacecraft() {

    }

    /\*\*

     \* Constructor with all fields

     \*

     \* @param name                the name of the spacecraft

     \* @param model               the model of the spacecraft

     \* @param engineType          the type of engine used by the spacecraft

     \* @param isRentable          the rentable status of the spacecraft

     \* @param rentalCostPerParsec the rental cost per parsec of the spacecraft

     \*/

    public BaseSpacecraft(String name, String model, String engineType, boolean isRentable,

            double rentalCostPerParsec) {

        this.name = name;

        this.model = model;

        this.engineType = engineType;

        this.isRentable = isRentable;

        this.rentalCostPerParsec = rentalCostPerParsec;

    }

    /\*\*

     \* Function to create a new BaseSpacecraft object based on user input

     \*

     \* @param scanner    the scanner to use for user input

     \* @param spacecraft the spacecraft to create

     \* @return the created spacecraft

     \*/

    protected static BaseSpacecraft createBaseSpacecraftFromInput(Scanner scanner, BaseSpacecraft spacecraft) {

        spacecraft.name = InputValidators.getInput(scanner, "Name: ");

        spacecraft.model = InputValidators.getInput(scanner, "Model: ");

        spacecraft.engineType = InputValidators.getInput(scanner, "Engine Type: ");

        spacecraft.isRentable = InputValidators.getYesNoInput(scanner, "Is Rentable (yes/no): ");

        spacecraft.rentalCostPerParsec = InputValidators.getDoubleInput(scanner, "Rental Cost Per Parsec: ");

        return spacecraft;

    }

    /\*\*

     \* Function to edit an existing BaseSpacecraft object based on user input

     \*

     \* @param scanner the scanner to use for user input

     \*/

    protected void editBaseSpacecraftFromInput(Scanner scanner) {

        String name = InputValidators.getEditInput(scanner, "Name");

        if (!name.isEmpty()) {

            this.name = name;

        }

        String model = InputValidators.getEditInput(scanner, "Model");

        if (!model.isEmpty()) {

            this.model = model;

        }

        String engineType = InputValidators.getEditInput(scanner, "Engine Type");

        if (!engineType.isEmpty()) {

            this.engineType = engineType;

        }

        String isRentableStr = InputValidators.getEditInput(scanner, "Is Rentable (y)es/(n)o");

        if (!isRentableStr.isEmpty()) {

            this.isRentable = InputValidators.convertYesNoToBoolean(isRentableStr);

        }

        String rentalCostPerParsecStr = InputValidators.getEditInput(scanner, "Rental Cost Per Parsec");

        if (!rentalCostPerParsecStr.isEmpty() && InputValidators.isValidDouble(rentalCostPerParsecStr)) {

            this.rentalCostPerParsec = Double.parseDouble(rentalCostPerParsecStr);

        }

    }

    /\*\*

     \* Function to get the crew capacity of the spacecraft

     \*

     \* @return the crew capacity of the spacecraft

     \*/

    public String getName() {

        return name;

    }

    /\*\*

     \* Function to set the crew capacity of the spacecraft

     \*

     \* @param name the crew capacity of the spacecraft

     \*/

    public void setName(String name) {

        this.name = name;

    }

    /\*\*

     \* Function to get the model of the spacecraft

     \*

     \* @return the model of the spacecraft

     \*/

    public String getModel() {

        return model;

    }

    /\*\*

     \* Function to set the model of the spacecraft

     \*

     \* @param model the model of the spacecraft

     \*/

    public void setModel(String model) {

        this.model = model;

    }

    /\*\*

     \* Function to get the type of engine used by the spacecraft

     \*

     \* @return the type of engine used by the spacecraft

     \*/

    public String getEngineType() {

        return engineType;

    }

    /\*\*

     \* Function to set the type of engine used by the spacecraft

     \*

     \* @param engineType the type of engine used by the spacecraft

     \*/

    public void setEngineType(String engineType) {

        this.engineType = engineType;

    }

    /\*\*

     \* Function to get the rentable status of the spacecraft

     \*

     \* @return the rentable status of the spacecraft

     \*/

    public boolean getIsRentable() {

        return isRentable;

    }

    /\*\*

     \* Function to set the rentable status of the spacecraft

     \*

     \* @return the rentable status of the spacecraft

     \*/

    public void setIsRentable(boolean isRentable) {

        this.isRentable = isRentable;

    }

    /\*\*

     \* Function to get the rentable status of the spacecraft

     \*

     \* @return the rentable status of the spacecraft

     \*/

    public double getRentalCostPerParsec() {

        return rentalCostPerParsec;

    }

    /\*\*

     \* Function to set the rentable status of the spacecraft

     \*

     \* @param isRentable the rentable status of the spacecraft

     \*/

    public void setRentalCostPerParsec(double rentalCostPerParsec) {

        this.rentalCostPerParsec = rentalCostPerParsec;

    }

    /\*\*

     \* Function to get the string representation of the spacecraft

     \*

     \* @return the string representation of the spacecraft

     \*/

    @Override

    public String toString() {

        return FormatUtils.padString("| Name: ", name + " |\n") +

                FormatUtils.padString("| Model: ", model + " |\n") +

                FormatUtils.padString("| Engine Type: ", engineType + " |\n") +

                FormatUtils.padString("| Is Rentable: ", InputValidators.convertBooleanToString(isRentable) + " |\n") +

                FormatUtils.padString("| Rental Cost Per Parsec: ", rentalCostPerParsec + " |\n");

    }

}

// ---------------------------------------------------------------------------

// models/Shuttle.java

package unit6.models;

import java.util.Scanner;

import unit6.interfaces.ShuttleSpacecraft;

import unit6.utilities.FormatUtils;

import unit6.utilities.InputValidators;

/\*\*

 \* Class for a shuttle

 \*/

public class Shuttle extends BaseSpacecraft implements ShuttleSpacecraft {

    private int crewCapacity;

    private String missionType;

    /\*\*

     \* Default constructor

     \*/

    public Shuttle() {

    }

    /\*\*

     \* Constructor with all fields

     \*

     \* @param name                the name of the shuttle

     \* @param model               the model of the shuttle

     \* @param engineType          the type of engine used by the shuttle

     \* @param isRentable          the rentable status of the shuttle

     \* @param rentalCostPerParsec the rental cost per parsec of the shuttle

     \* @param crewCapacity        the crew capacity of the shuttle

     \* @param missionType         the mission type of the shuttle

     \*/

    public Shuttle(String name, String model, String engineType, boolean isRentable, double rentalCostPerParsec,

            int crewCapacity, String missionType) {

        super(name, model, engineType, isRentable, rentalCostPerParsec);

        this.crewCapacity = crewCapacity;

        this.missionType = missionType;

    }

    /\*\*

     \* Function to get the type name of the shuttle

     \*

     \* @return the type name of the shuttle

     \*/

    public String getTypeName() {

        return "Shuttle";

    }

    /\*\*

     \* Function to create a new Shuttle object based on user input

     \*

     \* @param scanner the scanner to use for user input

     \* @return the created shuttle

     \*/

    public static Shuttle createShuttleFromInput(Scanner scanner) {

        System.out.println(FormatUtils.hr("=", " Enter Shuttle details "));

        Shuttle shuttle = (Shuttle) BaseSpacecraft.createBaseSpacecraftFromInput(scanner, new Shuttle());

        shuttle.crewCapacity = InputValidators.getIntInput(scanner, "Crew Capacity: ");

        shuttle.missionType = InputValidators.getInput(scanner, "Mission Type: ");

        return shuttle;

    }

    /\*\*

     \* Function to edit an existing Shuttle object based on user input

     \*

     \* @param scanner the scanner to use for user input

     \*/

    public void editFromInput(Scanner scanner) {

        System.out.println(FormatUtils.hr("=", " Edit Shuttle details "));

        super.editBaseSpacecraftFromInput(scanner);

        String crewCapacityStr = InputValidators.getEditInput(scanner, "Crew Capacity");

        if (!crewCapacityStr.isEmpty() && InputValidators.isValidInteger(crewCapacityStr)) {

            this.crewCapacity = Integer.parseInt(crewCapacityStr);

        }

        String missionType = InputValidators.getEditInput(scanner, "Mission Type");

        if (!missionType.isEmpty()) {

            this.missionType = missionType;

        }

    }

    /\*\*

     \* Method to get the crew capacity of the shuttle

     \*

     \* @return the crew capacity of the shuttle

     \*/

    @Override

    public int getCrewCapacity() {

        return crewCapacity;

    }

    /\*\*

     \* Method to set the crew capacity of the shuttle

     \*

     \* @param crewCapacity the crew capacity of the shuttle

     \*/

    @Override

    public void setCrewCapacity(int crewCapacity) {

        this.crewCapacity = crewCapacity;

    }

    /\*\*

     \* Method to get the mission type for the shuttle

     \*

     \* @return the mission type for the shuttle

     \*/

    @Override

    public String getMissionType() {

        return missionType;

    }

    /\*\*

     \* Method to set the mission type for the shuttle

     \*

     \* @param missionType the mission type for the shuttle

     \*/

    @Override

    public void setMissionType(String missionType) {

        this.missionType = missionType;

    }

    /\*\*

     \* Function to get the string representation of the shuttle

     \*

     \* @return the string representation of the shuttle

     \*/

    @Override

    public String toString() {

        return FormatUtils.hr("Shuttle") + "\n" +

                super.toString() +

                FormatUtils.padString("| Crew Capacity: ", crewCapacity + " |\n") +

                FormatUtils.padString("| Mission Type: ", missionType + " |\n") +

                FormatUtils.hr() + "\n";

    }

}

// ---------------------------------------------------------------------------

// models/Starfighter.java

package unit6.models;

import java.util.Scanner;

import unit6.interfaces.StarfighterSpacecraft;

import unit6.utilities.FormatUtils;

import unit6.utilities.InputValidators;

/\*\*

 \* Class for a starfighter

 \*/

public class Starfighter extends BaseSpacecraft implements StarfighterSpacecraft {

    private String weaponSystem;

    private boolean hyperdriveCapability;

    /\*\*

     \* Default constructor

     \*/

    public Starfighter() {

    }

    /\*\*

     \* Constructor with all fields

     \*

     \* @param name                 the name of the starfighter

     \* @param model                the model of the starfighter

     \* @param engineType           the type of engine used by the starfighter

     \* @param isRentable           the rentable status of the starfighter

     \* @param rentalCostPerParsec  the rental cost per parsec of the starfighter

     \* @param weaponSystem         the weapon system of the starfighter

     \* @param hyperdriveCapability the hyperdrive capability of the starfighter

     \*/

    public Starfighter(String name, String model, String engineType, boolean isRentable, double rentalCostPerParsec,

            String weaponSystem,

            boolean hyperdriveCapability) {

        super(name, model, engineType, isRentable, rentalCostPerParsec);

        this.weaponSystem = weaponSystem;

        this.hyperdriveCapability = hyperdriveCapability;

    }

    /\*\*

     \* Function to get the type name of the starfighter

     \*

     \* @return the type name of the starfighter

     \*/

    public String getTypeName() {

        return "Starfighter";

    }

    /\*\*

     \* Function to create a new Starfighter object based on user input

     \*

     \* @param scanner the scanner to use for user input

     \* @return the created starfighter

     \*/

    public static Starfighter createStarfighterFromInput(Scanner scanner) {

        System.out.println(FormatUtils.hr("=", " Enter Starfighter details "));

        Starfighter starfighter = (Starfighter) BaseSpacecraft.createBaseSpacecraftFromInput(scanner,

                new Starfighter());

        starfighter.weaponSystem = InputValidators.getInput(scanner, "Weapon System: ");

        starfighter.hyperdriveCapability = InputValidators.getYesNoInput(scanner, "Hyperdrive Capability (yes/no): ");

        return starfighter;

    }

    /\*\*

     \* Function to edit an existing Starfighter object based on user input

     \*

     \* @param scanner the scanner to use for user input

     \*/

    public void editFromInput(Scanner scanner) {

        System.out.println(FormatUtils.hr("=", " Edit Starfighter details "));

        super.editBaseSpacecraftFromInput(scanner);

        String weaponSystem = InputValidators.getEditInput(scanner, "Weapon System");

        if (!weaponSystem.isEmpty()) {

            setWeaponSystem(weaponSystem);

        }

        String hyperdriveCapabilityStr = InputValidators.getEditInput(scanner,

                "Hyperdrive Capability yes/no");

        if (!hyperdriveCapabilityStr.isEmpty()) {

            setHyperdriveCapability(InputValidators.convertYesNoToBoolean(hyperdriveCapabilityStr));

        }

    }

    /\*\*

     \* Method to get the weapon system of the starfighter

     \*

     \* @return the weapon system of the starfighter

     \*/

    @Override

    public String getWeaponSystem() {

        return weaponSystem;

    }

    /\*\*

     \* Method to set the weapon system of the starfighter

     \*

     \* @param weaponSystem the weapon system of the starfighter

     \*/

    @Override

    public void setWeaponSystem(String weaponSystem) {

        this.weaponSystem = weaponSystem;

    }

    /\*\*

     \* Method to get the hyperdrive capability of the starfighter

     \*

     \* @return the hyperdrive capability of the starfighter

     \*/

    @Override

    public boolean hasHyperdriveCapability() {

        return hyperdriveCapability;

    }

    /\*\*

     \* Method to set the hyperdrive capability of the starfighter

     \*

     \* @param hyperdriveCapability the hyperdrive capability of the starfighter

     \*/

    @Override

    public void setHyperdriveCapability(boolean hyperdriveCapability) {

        this.hyperdriveCapability = hyperdriveCapability;

    }

    /\*\*

     \* Method to get the details of the starfighter

     \*

     \* @return the details of the starfighter

     \*/

    @Override

    public String toString() {

        String hyperdriveStatus = hyperdriveCapability ? "Yes" : "No";

        return FormatUtils.hr("Starfighter") + "\n" +

                super.toString() +

                FormatUtils.padString("| Weapon System: ", weaponSystem + " |\n") +

                FormatUtils.padString("| Hyperdrive: ", hyperdriveStatus + " |\n") +

                FormatUtils.hr() + "\n";

    }

}

// ---------------------------------------------------------------------------

// models/CargoShip.java

package unit6.models;

import java.util.Scanner;

import unit6.interfaces.CargoShipSpacecraft;

import unit6.utilities.FormatUtils;

import unit6.utilities.InputValidators;

/\*\*

 \* Class for a cargo ship

 \*/

public class CargoShip extends BaseSpacecraft implements CargoShipSpacecraft {

    private double cargoCapacity;

    private boolean specializedCargoHandling;

    /\*\*

     \* Default constructor

     \*/

    public CargoShip() {

    }

    /\*\*

     \* Constructor with all fields

     \*

     \* @param name                     the name of the cargo ship

     \* @param model                    the model of the cargo ship

     \* @param engineType               the type of engine used by the cargo ship

     \* @param isRentable               the rentable status of the cargo ship

     \* @param rentalCostPerParsec      the rental cost per parsec of the cargo ship

     \* @param cargoCapacity            the cargo capacity of the cargo ship

     \* @param specializedCargoHandling the presence of a specialized cargo handling

     \*                                 system in the cargo ship

     \*/

    public CargoShip(String name, String model, String engineType, boolean isRentable, double rentalCostPerParsec,

            double cargoCapacity,

            boolean specializedCargoHandling) {

        super(name, model, engineType, isRentable, rentalCostPerParsec);

        this.cargoCapacity = cargoCapacity;

        this.specializedCargoHandling = specializedCargoHandling;

    }

    /\*\*

     \* Function to get the type name of the cargo ship

     \*

     \* @return the type name of the cargo ship

     \*/

    public String getTypeName() {

        return "CargoShip";

    }

    /\*\*

     \* Function to create a new CargoShip object based on user input

     \*

     \* @param scanner the scanner to use for user input

     \* @return the created cargo ship

     \*/

    public static CargoShip createCargoShipFromInput(Scanner scanner) {

        System.out.println(FormatUtils.hr("=", " Enter CargoShip details "));

        CargoShip cargoShip = (CargoShip) BaseSpacecraft.createBaseSpacecraftFromInput(scanner, new CargoShip());

        cargoShip.cargoCapacity = InputValidators.getDoubleInput(scanner, "Cargo Capacity: ");

        cargoShip.specializedCargoHandling = InputValidators.getYesNoInput(scanner,

                "Specialized Cargo Handling (yes/no): ");

        return cargoShip;

    }

    /\*\*

     \* Function to edit an existing CargoShip object based on user input

     \*

     \* @param scanner the scanner to use for user input

     \*/

    public void editFromInput(Scanner scanner) {

        System.out.println(FormatUtils.hr("=", " Edit CargoShip details "));

        super.editBaseSpacecraftFromInput(scanner);

        String cargoCapacityStr = InputValidators.getEditInput(scanner, "Cargo Capacity");

        if (!cargoCapacityStr.isEmpty() && InputValidators.isValidDouble(cargoCapacityStr)) {

            setCargoCapacity(Double.parseDouble(cargoCapacityStr));

        }

        String specializedHandlingStr = InputValidators.getEditInput(scanner, "Specialized Cargo Handling yes/no");

        if (!specializedHandlingStr.isEmpty()) {

            setSpecializedCargoHandling(InputValidators.convertYesNoToBoolean(specializedHandlingStr));

        }

    }

    /\*\*

     \* Method to get the cargo capacity of the cargo ship

     \*

     \* @return the cargo capacity of the cargo ship

     \*/

    @Override

    public double getCargoCapacity() {

        return cargoCapacity;

    }

    /\*\*

     \* Method to set the cargo capacity of the cargo ship

     \*

     \* @param cargoCapacity the cargo capacity of the cargo ship

     \*/

    @Override

    public void setCargoCapacity(double cargoCapacity) {

        this.cargoCapacity = cargoCapacity;

    }

    /\*\*

     \* Method to get the presence of a specialized cargo handling system in the

     \* cargo ship

     \*

     \* @return the presence of a specialized cargo handling system in the cargo ship

     \*/

    @Override

    public boolean hasSpecializedCargoHandling() {

        return specializedCargoHandling;

    }

    /\*\*

     \* Method to set the presence of a specialized cargo handling system in the

     \* cargo ship

     \*

     \* @param specializedCargoHandling the presence of a specialized cargo handling

     \*                                 system in the cargo ship

     \*/

    @Override

    public void setSpecializedCargoHandling(boolean specializedCargoHandling) {

        this.specializedCargoHandling = specializedCargoHandling;

    }

    /\*\*

     \* Function to get the string representation of the cargo ship

     \*

     \* @return the string representation of the cargo ship

     \*/

    @Override

    public String toString() {

        return FormatUtils.hr("CargoShip") + "\n" +

                super.toString() +

                FormatUtils.padString("| Cargo Capacity: ", cargoCapacity + " |\n") +

                FormatUtils.padString("| Specialized Handling: ",

                        InputValidators.convertBooleanToString(specializedCargoHandling) + " |\n")

                +

                FormatUtils.hr() + "\n";

    }

}

// ---------------------------------------------------------------------------

// utilities/FormatUtils.java

package unit6.utilities;

/\*\*

 \* Utility class for formatting strings

 \*/

public class FormatUtils {

    private static final int DEF\_LENGTH = 60;

    private static final String DEF\_HR\_CHAR = "-";

    private static final String DEF\_END\_CHAR = "|";

    private static final String DEF\_PAD\_CHAR = " ";

    /\*\*

     \* Function to pad a string to the left

     \*

     \* @param str    the string to pad

     \* @param length the length to pad to

     \* @return the padded string

     \*/

    public static String padRight(String str, int length) {

        return String.format("%1$-" + length + "s", str);

    }

    /\*\*

     \* Function to pad a string to the right

     \*

     \* @param str    the string to pad

     \* @param length the length to pad to

     \* @return the padded string

     \*/

    public static String padString(String prefix, String postfix) {

        return padString(prefix, postfix, DEF\_LENGTH);

    }

    /\*\*

     \* Function to pad a string to the right

     \*

     \* @param prefix      the prefix to pad

     \* @param postfix     the postfix to pad

     \* @param totalLength the length to pad to

     \* @return

     \*/

    public static String padString(String prefix, String postfix, int totalLength) {

        int contentLength = prefix.length() + postfix.length();

        if (contentLength <= totalLength) {

            int paddingLength = totalLength - contentLength;

            return prefix + String.format("%1$" + paddingLength + "s", "") + postfix;

        } else {

            int truncateLength = totalLength - prefix.length();

            postfix = postfix.substring(0, Math.min(postfix.length(), truncateLength));

            return prefix + postfix;

        }

    }

    /\*\*

     \* Function to pad a string to the right

     \*

     \* @param str    the string to pad

     \* @param length the length to pad to

     \* @return the padded string

     \*/

    public static String centerString(String middle) {

        return centerString(DEF\_END\_CHAR, middle, DEF\_END\_CHAR, DEF\_LENGTH);

    }

    /\*\*

     \* Function to pad a string to the right

     \*

     \* @param str    the string to pad

     \* @param length the length to pad to

     \* @return the padded string

     \*/

    public static String centerString(String prefix, String middle, String postfix, int totalLength) {

        int middleLength = totalLength - prefix.length() - postfix.length();

        if (middleLength < 0) {

            middle = middle.substring(0, Math.max(middle.length() + middleLength, 0));

            middleLength = middle.length();

        }

        int paddingTotal = middleLength - middle.length();

        int paddingSide = paddingTotal / 2;

        String padding = String.format("%1$" + paddingSide + "s", "");

        return prefix + padding + middle + padding + (paddingTotal % 2 != 0 ? DEF\_PAD\_CHAR : "") + postfix;

    }

    /\*\*

     \* Function to pad a string to the right

     \*

     \* @param str    the string to pad

     \* @param length the length to pad to

     \* @return the padded string

     \*/

    public static String hr() {

        return hr(DEF\_LENGTH, DEF\_HR\_CHAR, null);

    }

    /\*\*

     \* Function to pad a string to the right

     \*

     \* @param centeredString the string to pad

     \* @return the padded string

     \*/

    public static String hr(String centeredString) {

        return hr(DEF\_LENGTH, DEF\_HR\_CHAR, centeredString);

    }

    /\*\*

     \* Function to pad a string to the right

     \*

     \* @param length the length to pad to

     \* @return the padded string

     \*/

    public static String hr(int length) {

        return hr(length, DEF\_HR\_CHAR, null);

    }

    /\*\*

     \* Function to pad a string to the right

     \*

     \* @param length         the length to pad to

     \* @param centeredString the string to pad

     \* @return the padded string

     \*/

    public static String hr(String padding, int length) {

        return hr(length, padding, null);

    }

    /\*\*

     \* Function to pad a string to the right

     \*

     \* @param length         the length to pad to

     \* @param centeredString the string to pad

     \* @return the padded string

     \*/

    public static String hr(String padding, String centeredString) {

        return hr(DEF\_LENGTH, padding, centeredString);

    }

    /\*\*

     \* Function to pad a string to the right

     \*

     \* @param length         the length to pad to

     \* @param centeredString the string to pad

     \* @return the padded string

     \*/

    public static String hr(int length, String padding, String centeredString) {

        if (centeredString == null || centeredString.isEmpty()) {

            return String.format("%0" + length + "d", 0).replace("0", padding);

        } else {

            int totalPaddingLength = length - centeredString.length();

            int paddingSide = totalPaddingLength / 2;

            String prefix = String.format("%0" + paddingSide + "d", 0).replace("0", padding);

            String postfix = prefix;

            // Adjust for odd lengths

            if (totalPaddingLength % 2 != 0) {

                postfix += padding;

            }

            return prefix + centeredString + postfix;

        }

    }

}

// ---------------------------------------------------------------------------

// utilities/inputvalidators.java

package unit6.utilities;

import java.util.Scanner;

/\*\*

 \* Class for input validators

 \*/

public class InputValidators {

    private static final String DEF\_ERROR\_PREFIX = "Invalid input. Please enter ";

    /\*\*

     \* Function to check if a string is valid

     \*

     \* @param input the string to check

     \* @return true if the string is valid, false otherwise

     \*/

    public static boolean isValidString(String input) {

        return input != null && !input.isEmpty() && !input.trim().isEmpty();

    }

    /\*\*

     \* Function to check if an integer is valid

     \*

     \* @param input the integer to check

     \* @return true if the integer is valid, false otherwise

     \*/

    public static boolean isValidInteger(String input) {

        try {

            Integer.parseInt(input);

            return true;

        } catch (NumberFormatException e) {

            return false;

        }

    }

    /\*\*

     \* Function to check if a double is valid

     \*

     \* @param input the double to check

     \* @return true if the double is valid, false otherwise

     \*/

    public static boolean isValidDouble(String input) {

        try {

            Double.parseDouble(input);

            return true;

        } catch (NumberFormatException e) {

            return false;

        }

    }

    /\*\*

     \* Function to get a yes/no input from the user

     \*

     \* @param scanner the scanner to use for user input

     \* @param prompt  the prompt to display to the user

     \* @return true if the user input is yes, false otherwise

     \*/

    public static boolean getYesNoInput(Scanner scanner, String prompt) {

        String input = getInput(scanner, prompt);

        while (!isValidYesNoResponse(input) && !input.isEmpty()) {

            System.out.println(DEF\_ERROR\_PREFIX + "(y)es/(n)o:");

            input = getInput(scanner, prompt);

        }

        return input.equalsIgnoreCase("yes") || input.toLowerCase().startsWith("y");

    }

    /\*\*

     \* Function to get a yes/no input from the user

     \*

     \* @param scanner the scanner to use for user input

     \* @return true if the user input is yes, false otherwise

     \*/

    public static boolean isValidYesNoResponse(String input) {

        return input.equalsIgnoreCase("yes") ||

                input.equalsIgnoreCase("no") ||

                input.equalsIgnoreCase("y") ||

                input.equalsIgnoreCase("n");

    }

    /\*\*

     \* Function to convert a yes/no string to a boolean

     \*

     \* @param input the string to convert

     \* @return true if the string is yes, false otherwise

     \*/

    public static boolean convertYesNoToBoolean(String input) {

        return input.equalsIgnoreCase("yes") || input.equalsIgnoreCase("y");

    }

    /\*\*

     \* Function to convert a boolean to a yes/no string

     \*

     \* @param input the boolean to convert

     \* @return "yes" if the boolean is true, "no" otherwise

     \*/

    public static String convertBooleanToString(boolean input, String trueString, String falseString) {

        return input ? trueString : falseString;

    }

    /\*\*

     \* Function to convert a boolean to a yes/no string

     \*

     \* @param input the boolean to convert

     \* @return "yes" if the boolean is true, "no" otherwise

     \*/

    public static String convertBooleanToString(boolean input) {

        return convertBooleanToString(input, "yes", "no");

    }

    /\*\*

     \* Function to get a double input from the user

     \*

     \* @param scanner the scanner to use for user input

     \* @param prompt  the prompt to display to the user

     \* @return the double input from the user

     \*/

    public static double getDoubleInput(Scanner scanner, String prompt) {

        System.out.print(prompt);

        String input = scanner.nextLine();

        while (!isValidDouble(input) && !input.isEmpty()) {

            System.out.println(DEF\_ERROR\_PREFIX + "a valid number:");

            System.out.print(prompt);

            input = scanner.nextLine();

        }

        return input.isEmpty() ? 0.0 : Double.parseDouble(input);

    }

    /\*\*

     \* Function to get a double input from the user

     \*

     \* @param scanner the scanner to use for user input

     \* @param prompt  the prompt to display to the user

     \* @return the double input from the user

     \*/

    public static String getEditInput(Scanner scanner, String preString, String prompt, String postString) {

        return getInput(scanner, preString + prompt + postString);

    }

    /\*\*

     \* Function to get a double input from the user

     \*

     \* @param scanner the scanner to use for user input

     \* @param prompt  the prompt to display to the user

     \* @return the double input from the user

     \*/

    public static String getEditInput(Scanner scanner, String prompt) {

        return getEditInput(scanner, "New ", prompt, " (leave blank to keep current): ");

    }

    /\*\*

     \* Function to get a string input from the user

     \*

     \* @param scanner the scanner to use for user input

     \* @param prompt  the prompt to display to the user

     \* @return the string input from the user

     \*/

    public static String getInput(Scanner scanner, String prompt) {

        System.out.print(prompt);

        String input = scanner.nextLine();

        while (!isValidString(input)) {

            System.out.println(DEF\_ERROR\_PREFIX + "a valid string:");

            System.out.print(prompt);

            input = scanner.nextLine();

        }

        return input;

    }

    /\*\*

     \* Function to get an integer input from the user

     \*

     \* @param scanner the scanner to use for user input

     \* @param prompt  the prompt to display to the user

     \* @return the integer input from the user

     \*/

    public static int getIntInput(Scanner scanner, String prompt) {

        System.out.print(prompt);

        String input = scanner.nextLine();

        while (!isValidInteger(input) && !input.isEmpty()) {

            System.out.println(DEF\_ERROR\_PREFIX + "a valid integer:");

            System.out.print(prompt);

            input = scanner.nextLine();

        }

        return input.isEmpty() ? 0 : Integer.parseInt(input);

    }

}

// ---------------------------------------------------------------------------

// utilities/SampleData.java

package unit6.utilities;

import unit6.models.Starfighter;

import unit6.models.Shuttle;

import unit6.interfaces.Spacecraft;

import unit6.models.CargoShip;

import java.util.ArrayList;

/\*\*

 \* Utility class to generate sample data

 \*/

public class SampleData {

        /\*\*

         \* Method to generate a random double between min and max

         \*

         \* @param min the minimum value

         \* @param max the maximum value

         \* @return a random double between min and max

         \*/

        private static double RandomDouble(double min, double max) {

                return (Math.random() \* (max - min)) + min;

        }

        /\*\*

         \* Method to generate a random double between 50 and 100

         \*

         \* @param max the maximum value

         \* @return a random double between 50 and 100

         \*/

        private static int RandomDouble() {

                return (int) RandomDouble(50, 100);

        }

        /\*\*

         \* Method builds and returns a list of all sample spacecrafts

         \* there are 3 types of spacecrafts: Starfighter, Shuttle, and CargoShip

         \* 3 sample spacecrafts of each type are created

         \*

         \* @return a list of all sample spacecrafts

         \*/

        public static void getAllSamples(ArrayList<Spacecraft> list) {

                list.add(new Starfighter(

                                "X-Wing",

                                "T-65",

                                "Incom 4L4 Fusial Thrust Engine",

                                true,

                                RandomDouble(),

                                "Laser Cannons",

                                true));

                list.add(new Starfighter("TIE Fighter",

                                "Twin Ion Engine LN Starfighter",

                                "Sienar Fleet Systems",

                                true,

                                RandomDouble(),

                                "Laser Cannons",

                                false));

                list.add(new Starfighter("A-Wing",

                                "RZ-1",

                                "Alliance Underground Engineering",

                                true,

                                RandomDouble(),

                                "Laser Cannons",

                                true));

                list.add(new Shuttle("Galaxy Class",

                                "NCC-1701-D",

                                "Warp Drive",

                                true,

                                RandomDouble(),

                                1014,

                                "Exploration and Diplomacy"));

                list.add(new Shuttle("Imperial Shuttle",

                                "Lambda-class T-4a",

                                "Cygnus Spaceworks",

                                true,

                                RandomDouble(),

                                20,

                                "Troop Transport"));

                list.add(new Shuttle("Runabout",

                                "Danube Class",

                                "Warp Drive",

                                true,

                                RandomDouble(),

                                40,

                                "Short-range Missions"));

                list.add(new CargoShip("Nostromo",

                                "Lockmart CM-88B Bison M-Class Starfreighter",

                                "Rolls-Royce N66 Cyclone Thrust Tunnels",

                                true,

                                RandomDouble(),

                                20000,

                                true));

                list.add(new CargoShip("Sulaco",

                                "Conestoga-class Troop Transport",

                                "Westingland A24",

                                true,

                                RandomDouble(),

                                50000,

                                true));

                list.add(new CargoShip("Valley Forge",

                                "American Airlines Space Freighter",

                                "General Electric CF6-50",

                                true,

                                RandomDouble(),

                                100000,

                                false));

        }

}

// ---------------------------------------------------------------------------

Output:

------------------------------------------------------------

| Intergalactic Voyages: Simon's Starship Bazaar |

| Zooming Through the Cosmos at Warp Speed! |

| Where Space Oddities Become Your Reality |

| \* All sales are final. No refunds or exchanges \* |

------------------------------------------------------------

======================== Main Menu =========================

1. Add Spacecraft

2. Edit Spacecraft

3. Delete Spacecraft

4. List All Spacecrafts (Detailed)

5. List All Spacecrafts (Simple)

E. Exit

Enter your choice: 1

==================== Select type to add ====================

1. Starfighter

2. Shuttle

3. Cargo Ship

B. Back to Main Menu

Enter your choice: 1

================ Enter Starfighter details =================

Name: Millennium Falcon

Model: YT-1300 Corellian Freighter

Engine Type: Girodyne SRB42 Sublight Engines

Is Rentable (yes/no): yes

Rental Cost Per Parsec: 456.78

Weapon System: Laser Cannons, Concussion Missiles

Hyperdrive Capability (yes/no): y

Spacecraft added successfully!

======================== Main Menu =========================

1. Add Spacecraft

2. Edit Spacecraft

3. Delete Spacecraft

4. List All Spacecrafts (Detailed)

5. List All Spacecrafts (Simple)

E. Exit

Enter your choice: 2

=========================== Edit ===========================

Select a spacecraft to edit:

1. X-Wing (Starfighter)

2. TIE Fighter (Starfighter)

3. A-Wing (Starfighter)

4. Galaxy Class (Shuttle)

5. Imperial Shuttle (Shuttle)

6. Runabout (Shuttle)

7. Nostromo (CargoShip)

8. Sulaco (CargoShip)

9. Valley Forge (CargoShip)

10. Millennium Falcon (Starfighter)

B. Back to Main Menu

Enter your choice: 1

================= Edit Starfighter details =================

New Name (leave blank to keep current):

New Model (leave blank to keep current):

New Engine Type (leave blank to keep current):

New Is Rentable (y)es/(n)o (leave blank to keep current): n

New Rental Cost Per Parsec (leave blank to keep current):

New Weapon System (leave blank to keep current):

New Hyperdrive Capability yes/no (leave blank to keep current):

======================== Main Menu =========================

1. Add Spacecraft

2. Edit Spacecraft

3. Delete Spacecraft

4. List All Spacecrafts (Detailed)

5. List All Spacecrafts (Simple)

E. Exit

Enter your choice: 3

========================== Delete ==========================

Select a spacecraft to delete:

1. X-Wing (Starfighter)

2. TIE Fighter (Starfighter)

3. A-Wing (Starfighter)

4. Galaxy Class (Shuttle)

5. Imperial Shuttle (Shuttle)

6. Runabout (Shuttle)

7. Nostromo (CargoShip)

8. Sulaco (CargoShip)

9. Valley Forge (CargoShip)

10. Millennium Falcon (Starfighter)

B. Back to Main Menu

Enter your choice: 2

Are you sure you want to delete the Starfighter 'TIE Fighter'? (yes/no)

yes

Starfighter deleted successfully!

======================== Main Menu =========================

1. Add Spacecraft

2. Edit Spacecraft

3. Delete Spacecraft

4. List All Spacecrafts (Detailed)

5. List All Spacecrafts (Simple)

E. Exit

Enter your choice: 4

================= List of All Spacecrafts ==================

======================= Starfighters =======================

------------------------Starfighter-------------------------

| Name: X-Wing |

| Model: T-65 |

| Engine Type: Incom 4L4 Fusial Thrust Engine |

| Is Rentable: no |

| Rental Cost Per Parsec: 89.0 |

| Weapon System: Laser Cannons |

| Hyperdrive: Yes |

------------------------------------------------------------

------------------------Starfighter-------------------------

| Name: A-Wing |

| Model: RZ-1 |

| Engine Type: Alliance Underground Engineering |

| Is Rentable: yes |

| Rental Cost Per Parsec: 55.0 |

| Weapon System: Laser Cannons |

| Hyperdrive: Yes |

------------------------------------------------------------

------------------------Starfighter-------------------------

| Name: Millennium Falcon |

| Model: YT-1300 Corellian Freighter |

| Engine Type: Girodyne SRB42 Sublight Engines |

| Is Rentable: yes |

| Rental Cost Per Parsec: 456.78 |

| Weapon System: Laser Cannons, Concussion Missiles |

| Hyperdrive: Yes |

------------------------------------------------------------

========================= Shuttles =========================

--------------------------Shuttle---------------------------

| Name: Galaxy Class |

| Model: NCC-1701-D |

| Engine Type: Warp Drive |

| Is Rentable: yes |

| Rental Cost Per Parsec: 78.0 |

| Crew Capacity: 1014 |

| Mission Type: Exploration and Diplomacy |

------------------------------------------------------------

--------------------------Shuttle---------------------------

| Name: Imperial Shuttle |

| Model: Lambda-class T-4a |

| Engine Type: Cygnus Spaceworks |

| Is Rentable: yes |

| Rental Cost Per Parsec: 93.0 |

| Crew Capacity: 20 |

| Mission Type: Troop Transport |

------------------------------------------------------------

--------------------------Shuttle---------------------------

| Name: Runabout |

| Model: Danube Class |

| Engine Type: Warp Drive |

| Is Rentable: yes |

| Rental Cost Per Parsec: 59.0 |

| Crew Capacity: 40 |

| Mission Type: Short-range Missions |

------------------------------------------------------------

======================= Cargo Ships ========================

-------------------------CargoShip--------------------------

| Name: Nostromo |

| Model: Lockmart CM-88B Bison M-Class Starfreighter |

| Engine Type: Rolls-Royce N66 Cyclone Thrust Tunnels |

| Is Rentable: yes |

| Rental Cost Per Parsec: 80.0 |

| Cargo Capacity: 20000.0 |

| Specialized Handling: yes |

------------------------------------------------------------

-------------------------CargoShip--------------------------

| Name: Sulaco |

| Model: Conestoga-class Troop Transport |

| Engine Type: Westingland A24 |

| Is Rentable: yes |

| Rental Cost Per Parsec: 64.0 |

| Cargo Capacity: 50000.0 |

| Specialized Handling: yes |

------------------------------------------------------------

-------------------------CargoShip--------------------------

| Name: Valley Forge |

| Model: American Airlines Space Freighter |

| Engine Type: General Electric CF6-50 |

| Is Rentable: yes |

| Rental Cost Per Parsec: 72.0 |

| Cargo Capacity: 100000.0 |

| Specialized Handling: no |

------------------------------------------------------------

======================== Main Menu =========================

1. Add Spacecraft

2. Edit Spacecraft

3. Delete Spacecraft

4. List All Spacecrafts (Detailed)

5. List All Spacecrafts (Simple)

E. Exit

Enter your choice: 5

============= List of All Spacecrafts (Simple) =============

Starfighter - X-Wing

Starfighter - A-Wing

Shuttle - Galaxy Class

Shuttle - Imperial Shuttle

Shuttle - Runabout

CargoShip - Nostromo

CargoShip - Sulaco

CargoShip - Valley Forge

Starfighter - Millennium Falcon

======================== Main Menu =========================

1. Add Spacecraft

2. Edit Spacecraft

3. Delete Spacecraft

4. List All Spacecrafts (Detailed)

5. List All Spacecrafts (Simple)

E. Exit

Enter your choice: E

Thank you for using Simon's Starship Bazaar!

Goodbye. Live long and prosper. NO REFUNDS.

The main parts of the code:

1. MainSystem Class (MainSystem.java): This is the main class of the application. It includes the main method, which serves as the entry point for the program. It handles user interactions, displays the main menu, and coordinates the addition, editing, deletion, and listing of spacecraft. The class uses an ArrayList<Spacecraft> to store the spacecraft collection and a Scanner for user input.
2. Interfaces (in the interfaces package):
   1. Spacecraft Interface: Defines the basic structure and methods common to all types of spacecraft, such as getters and setters for name, model, engine type, rental status, and rental cost. It also includes the editFromInput function for editing spacecraft details.
   2. ShuttleSpacecraft Interface: Extends the Spacecraft interface, adding methods specific to shuttles, like crew capacity and mission type.
   3. StarfighterSpacecraft Interface: Similar to ShuttleSpacecraft, but for starfighters, including methods for weapon systems and hyperdrive capability.
   4. CargoShipSpacecraft Interface: Customized for cargo ships, including methods for cargo capacity and specialized handling.
3. Model Classes:
   1. BaseSpacecraft Class: An abstract class that implements common attributes and methods shared by all spacecraft types. It contains properties like name, model, engine type, rental status, and cost.
   2. Shuttle, Starfighter, CargoShip Classes: These classes extend BaseSpacecraft and implement their respective interfaces. They add specific properties and methods unique to each type of spacecraft, such as the create method for creating new instances and editFromInput for modifying existing ones.
4. Utility Classes:
   1. FormatUtils Class: Provides static methods for string formatting used across the application, such as padding strings, centering text, and creating horizontal lines for UI separation.
   2. InputValidators Class: Offers static methods for validating user input and parsing different data types, ensuring the robustness of user interactions.
   3. SampleData Class: Generates a collection of sample spacecraft data for demonstration purposes. This includes various types of spacecraft with pre-defined attributes to populate the application initially.
5. This system is designed with object-oriented principles, ensuring encapsulation and separation of concerns. Each class and interface has a distinct role, contributing to the modularity and maintainability of the codebase. The program's user interface is text-based, navigated through a console using standard input and output.

## References

Java Language and Virtual Machine Specifications

* <https://docs.oracle.com/javase/specs/index.html>

Introduction to Programming Using Java - Version 9.0, JavaFX Edition

* <https://math.hws.edu/javanotes/>

Star Wars Franchise

* Lucas, G. (Creator). (1977-present). Star Wars [Film Series]. Lucasfilm Ltd.
* Wookieepedia, Star Wars Fan Wiki, <https://starwars.fandom.com/>

Star Trek Franchise

* + Roddenberry, G. (Creator). (1987-1994). Star Trek: The Next Generation [Television Series]. Paramount Pictures.
  + Berman, R., & Piller, M. (Creators). (1993-1999). Star Trek: Deep Space Nine [Television Series]. Paramount Pictures.
  + Memory Alpha, Star Trek Fan Wiki, <https://memory-alpha.fandom.com/>

Alien Film Franchise

* + Scott, R. (Director). (1979). Alien [Film]. 20th Century Fox.
  + Cameron, J. (Director). (1986). Aliens [Film]. 20th Century Fox.
  + Xenopedia, Alien Franchise Fan Wiki, <https://avp.fandom.com/>

Silent Running Film

* + Trumbull, D. (Director). (1972). Silent Running [Film]. Universal Pictures.
  + Sci-Fi Movie Wiki, <https://movies.fandom.com/>

**Disclaimer**: All names and terms that are trademarks are used here for educational and illustrative purposes only, without any intent of infringement. All rights to these trademarks are reserved by their respective owners.

**Source Code:**

