



UNIVERSITI TEKNOLOGI MALAYSIA

OBJECT ORIENTED PROGRAMMING

(SECJ2154)

SEMESTER 2 2023/2024

GROUP PROJECT

CAMPING TRIP PLANNER

MEGAT MUHAMMAD ZAFRAN BIN MEGAT MUAZZAM A22EC0194

DANIAL ERFAN SHAH BIN NOR AZAM SHAH A22EC0151

MUHAMMAD ARIF FIKRY BIN NOOR KHARIZAN A22EC0203

KUGANRAJ A/L RAMESH A22EC0177

NURAINN SOFEA BINTI NIZAMIL FAIRUZ A23CS5050

04-SECJ2154

SECTION 04

Lecturer:

MADAM LIZAWATI MI YUSUF

18th JUNE 2024

SECTION A: PROJECT DESCRIPTION

The work in this page has been done by : NURAINN SOFEA BINTI NIZAMIL FAIRUZ
Date:20/6/2024

Problem Analysis: Planning a camping trip has several steps to ensure safety, enjoyment, and organization. These steps include choosing a destination, choosing a camping style, planning meals and snacks, preparing clothing and personal items, planning activities and entertainment plans. Due to these tedious jobs, it is quite hard to keep track of all of this planning information that is quite essential to ensure a seamlessly smooth camping event.

Apart from that using a command-line interface for an app is quite dull and sometimes confusing for users to use, therefore having a graphical like interface is quite important to ensure the user experience is fulfilled.

The work in this page has been done by : DANIAL ERFAN SHAH BIN NOR AZAM SHAH
Date:20/6/2024

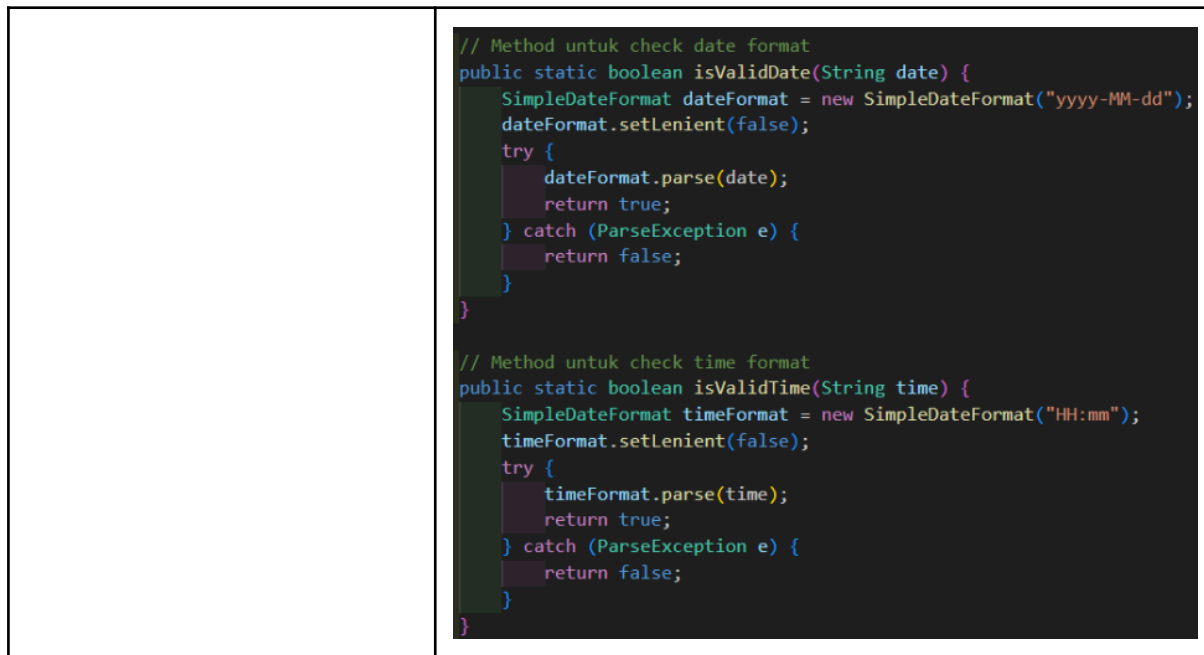
Synopsis:

General Concept: The Camping Trips Planner is an application designed to help users plan and manage their camping trips efficiently. The system allows users to input all details about their trips, including names, locations, budgets, activities, and dates and times. By integrating these details, the system provides a comprehensive overview of each trip, helping users organize their plans and budgets effectively.

Key Features:

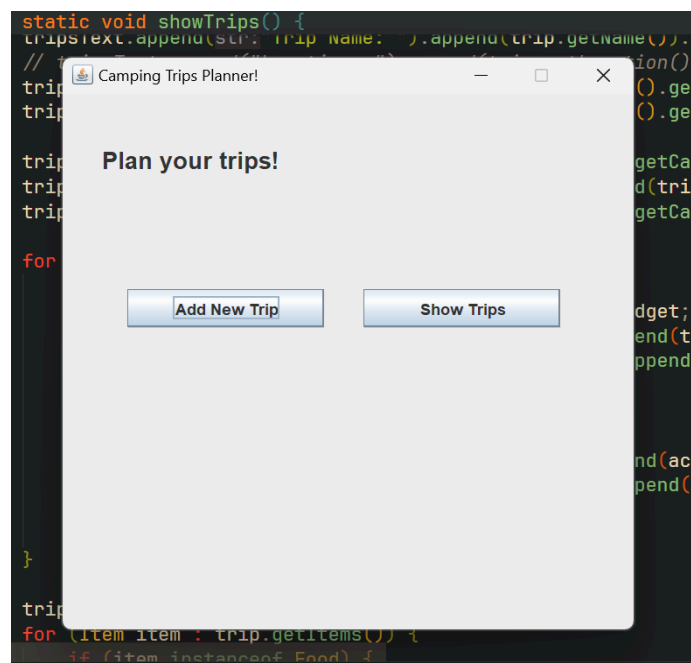
Encapsulation and Data Hiding	<p>Trip details are encapsulated within appropriate classes like, CampingTrips, Activities with private access modifiers, ensuring data security and integrity.</p> <pre>public class Activities extends Budget { private String activityName; private double activityCost; private Color activityColor; } class CampingTrips { private String name; private DateAndTime dateTime; private Vector<Budget> budget = new Vector<Budget>(); private Vector<Item> item = new Vector<Item>(); private Location campground = new Campground(); }</pre>
Association (Composition and Aggregation)	

	<p>Utilizes composition to encapsulate Location within Init, where activities are integral parts of each trip.</p> <div><div><pre>class CampingTrips { - name: String - dateTime: DateAndTime - budget: Vector<Budget> - item: Vector<Item> - campground: Campground + CampingTrips(name: String, dateTime: DateAndTime, tripBudgets: Vector<Budget>, item: Vector<Item>, String campName, String campDescription, int campSites) + getCampground(): Campground + getDateTime(): DateAndTime }</pre></div><div><pre>class Budget { - sumBudget double - sumUsed double + Budget() + setBudget(sumBudget : double): void + getBudget(): double + setUsed(sumUsed : double): void + getUsed(): double + addExpense(amount : double): void + getRemainingBudget(): double + displayBudget(): void }</pre></div></div> <p>Employs aggregation to manage budgets (Budget class) associated with trips, allowing flexible budgeting for activities and transportation.</p>
Inheritance and Polymorphism	<p>Implements inheritance with the Budget class serving as the parent class for Transportation and Activities, enabling code reuse and hierarchy in budget management.</p> <p>Utilizes polymorphism to handle different types of budgets uniformly, enhancing flexibility in budget calculations and management.</p> <div><div><pre>class Budget { - sumBudget double - sumUsed double + Budget() + setBudget(sumBudget : double): void + getBudget(): double + setUsed(sumUsed : double): void + getUsed(): double + addExpense(amount : double): void + getRemainingBudget(): double + displayBudget(): void }</pre></div><div><pre>class Activity { - activityName: String - activityCost: double - activityColor: Color + Activities() + Activities(activityName: String, activityCost: double, activityColor: Color) + setName(activityName: String): void + getName(): String + setCost(activityCost: double): void + getCost(): double + setColor(activityColor: Color): void + getColor(): Color + setActivityDetails(activityName: String, activityCost: double, activityColor: Color): void + addActivityExpense(): void + displayBudget(): void }</pre></div><div><pre>class Transportation { - transportType String - transportCost double + Transportation() + Transportation(transportType : String, transportCost : double) + setName(transportType : String) void + getName() String + setCost(transportCost : double) void + getCost() double + setTransportDetails(transportType : String, transportCost : double) + addTransportExpense() void + displayBudget() void }</pre></div></div>
Exception Handling	<p>Implements exception handling mechanisms like ParseException in DateAndTime.java to manage and recover from errors related to input validation and date/time formatting.</p>



How to Use the System:

1. Main Menu:



When launching the application, users are greeted with the main menu offering options to add new trips or view existing ones.

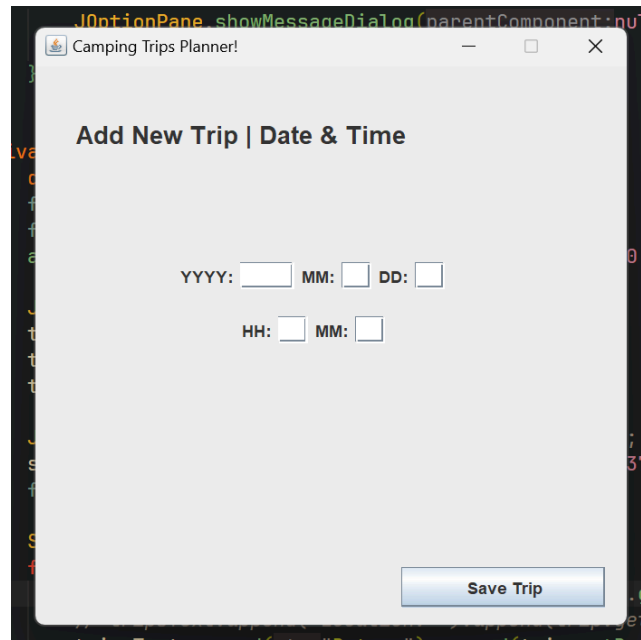
2. Adding a New Trip:

Users can click to the "Add New Trip" button where user put in data of the trip's name and location. They will also specify transportation details and associated costs.

Next, users will go to add activities for the trip, include activity names, costs, and color-coded categories.

Then, user need to add Food that will be bring for trip where user enter the detail of food with it quantity and also its expired date.

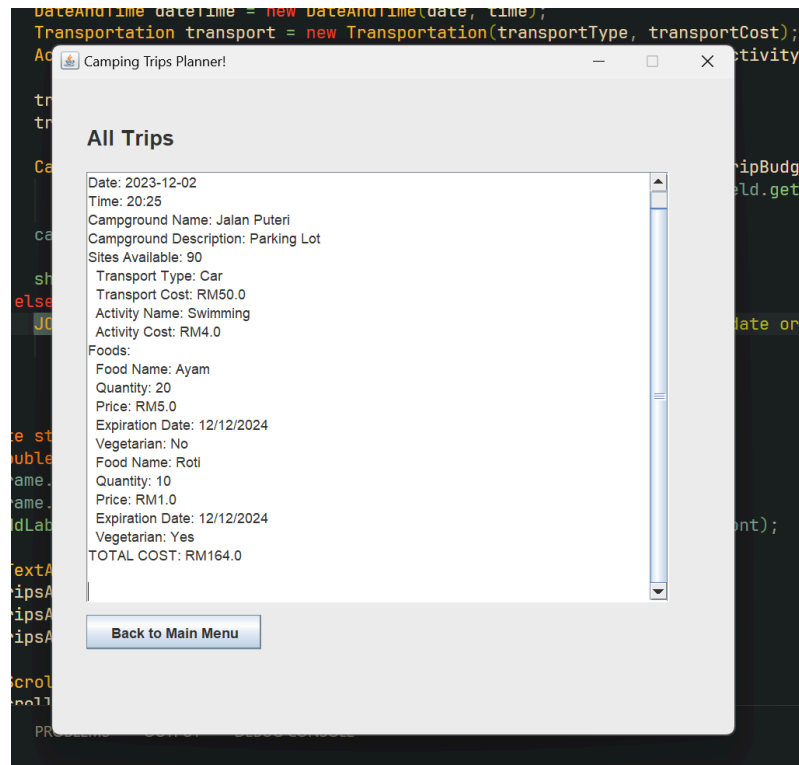
3. Setting Date and Time:



The screenshot shows a Java Swing window titled "Camping Trips Planner!". Inside the window is a dialog box titled "Add New Trip | Date & Time". The dialog contains two rows of input fields. The first row is for the date, with labels "YYYY:", "MM:", and "DD:" followed by single-digit text boxes. The second row is for the time, with labels "HH:" and "MM:" followed by single-digit text boxes. A "Save Trip" button is located at the bottom right of the dialog. The background of the image shows a dark IDE with some code snippets visible.

After entering the trip's activities, users will need to give the date and time of their trip, make sure that the schedule is accurately planned.

4. Viewing Saved Trips & Displaying Trip Details:



Users can access the "Show Trips" option from the main menu to see a list of all saved trips, include detail information about transportation, activities, and costs.

The system displays details of each trip, include total costs and budget breakdowns, help users review and manage their plans smoothly.

Objective and Scope:

Objective:

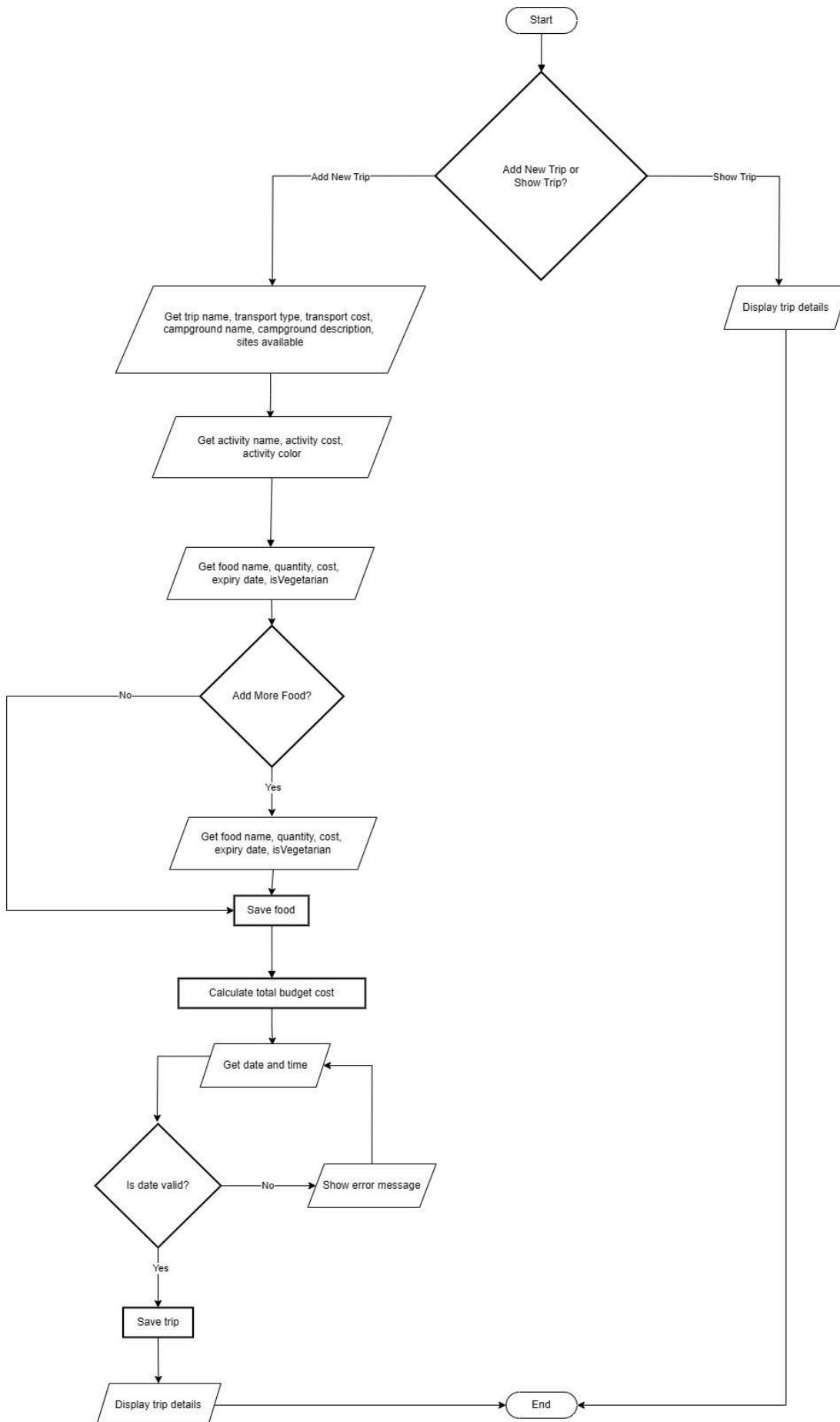
The objective of the Camping Trips Planner is to offer a user-friendly and systematic tool for organize camping trips. Through combination of trip details, budget management, and scheduling, the system focus to help users in comprehensive trip planning while minimize potential difficulty.

To enhance user experience, the system will feature a user-friendly interface with pop-up windows for easier to navigate and clear to understand. Each function will be accessible through dedicated windows to facilitate straightforward tracking and prevent user confuse, ensure efficient usage of the system's functionalities.

Scope:

Trip Management	<ul style="list-style-type: none">• Users can create, view, and manage multiple camping trips.• Each trip includes details such as the trip name, location, date, and time and more.
Budget Management	<ul style="list-style-type: none">• Allows users to specify and managing the budget for each trip.• Users can add many budget items, include transportation and activities, and track the total and remaining budget.
Activity Planning	<ul style="list-style-type: none">• Users can plan the activities for the trips, specifying details like activity names, costs, and color-coded categories.• The system integrates these activities into the overall trip plan, helping users organize their itineraries.
Food Management	<ul style="list-style-type: none">• Users can enter and track food item details such as name, quantity, price, expiration date, and vegetarian status.• Specify and categorize food items based on vegetarian status to accommodate dietary preferences.• Add and manage quantities of food items to keep an accurate inventory and avoid shortages.
Date and Time Scheduling	<ul style="list-style-type: none">• Users can specify the date and time for their trips, ensuring accurate scheduling.• The system validates the date and time inputs to prevent errors and conflicts in the trip schedule.
User Interface	<ul style="list-style-type: none">• Offers a graphical user interface for easy navigation and input.• Displays comprehensive trip details and budget summaries, enhancing user experience.

Flow Chart:



Workflow:

1. Firstly, after starting the program, ask user to choose either **“Add New Trip”** or **“Show Trip”**
2. If **“Add New Trip”**, **get the trip name, transportation type, cost and campground name, description and sites available** from the user.
3. Then, **get the activity name, cost, color** from the user too.
4. Then, **get the food name, cost, expiry date, isVegetarian** from the user.
5. Plus, ask the user if they want to add food. If **“Add Food”**, **add the quantity of the food**, then get the food details. If false, **save the food** that was entered by the user.
6. **Calculate the total cost** by adding the activity cost, transport cost and food cost.
7. Then, **get the date and time** of the camping trip from the user.
8. If the date entered is valid, save the trip details. **If not valid, show an error message** and ask the user to enter the date and time again correctly.
9. After saving the trip, **display the trip details**.
10. If **“Show Trip”**, **display the saved trip details**.
11. End.

OO Concepts:

1. ENCAPSULATION & DATA HIDING

```
public class Budget {
    private double sumBudget;
    private double sumUsed;

    public Budget() {
        this.sumBudget = 0;
        this.sumUsed = 0;
    }

    public void setBudget(double sumBudget) {
        this.sumBudget = sumBudget;
    }

    public double getBudget() {
        return sumBudget;
    }

    public void setUsed(double sumUsed) {
        this.sumUsed = sumUsed;
    }

    public double getUsed() {
        return sumUsed;
    }
}
```

Encapsulation and Data hiding concepts exist in nearly all of the classes in your project but to make it simple, we will use the class BUDGET. For some context, Encapsulation in java means to integrate data and code into a single unit and Data Hiding refers to encapsulating data within a class and restricting some attributes and methods using access modifiers.

For our case, in class Budget we can see the keyword “Private” where it hides the attribute sumBudget and sumUsed from the method in the class, this is due to the fact that we want to protect the attribute so that we will not abuse it by directly calling it with a “Public” type. This method is called encapsulation as we are encapsulating the attributes into one container and not letting outsiders disturb it.

For Data Hiding, we can refer to the “**public double getBudget**” and “**public double getUsed**” as getters or mutators that makes us able to access the private attribute by returning the value that is associated with the type of return statement. This concept is important for a large code just like this project as we want to protect the attributes from unauthorised data change and it also makes the code more rigid and secure .

2. AGGREGATION & COMPOSITION

```
class CampingTrips {
    private String name;
    private DateAndTime dateTime;
    private Vector<Budget> budget = new Vector<Budget>();
    private Vector<Item> item = new Vector<Item>();
    private Location campground = new Campground();

    public CampingTrips(String name, DateAndTime dateTime,
        Vector<Budget> tripBudgets, Vector<Item> item,
        String campName, String campDescription,
        int campSites) {
        this.name = name;
        this.dateTime = dateTime;
        this.budget = tripBudgets;
        this.item = item;
        this.campground = new Campground(campName, campDescription, campSites);
    }
};
```

In our code we have implemented some class relationships which are Aggregation and Composition. Aggregation refers to a relationship between two classes that have some sort of ownership relationship, which is mainly an “has-a” relationship. This means that for a class that has an aggregation relationship, when the object is deleted, it will affect the main class as we are only passing the value of an object. Composition on the other hand is a relationship that is “Exclusive owned” by an object as they have a “contain a” relationship between the classes. Composition because they are passing an object unlike a value will affect the main class if it is deleted.

As an example, in the class `CampingTrips` we can see the attributes being the other classes that have a relationship with this particular class. If we look closer, all attributes are creating an object based on the class such as “**private DateAndTime dateTime**” or “**private Vector<Item> item = new Vector<Item>**”. These objects are used in our code to call the methods that are available in the class into our main to use.

Aggregation can be seen in the Constructor name “**private CampingTrips**” and the attributes inside the bracket are what separates aggregation and composition as aggregation will need us to have the value of the class object in its parameter because we need its value and we can see it with “**DateAndTime dateTime**” with the method “**this.dateTime = dateTime**”.

Composition on the other hand does not exist in the parameter of the Constructor but it will create an object inside the method just like “**this.campground = new Campground(campName, campDescription, campSites)**”. This is because composition will create an object for every iteration of input that exists.

3. INHERITANCE

```
class Item {
    private String name;
    private int quantity;
    private double price;
    private String description;

    public Item() {
    }

    public Item(String name, int quantity, double price) {
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }

    public String getName() {
        return name;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    public double getPrice() {
        return price;
    }

    public String getDescription() { // description of Item
        return description;
    }

    public double calcTotalCost() { // calculate price of any item
        return price * quantity;
    }
}
```

```
public class Food extends Item {
    private String expirationDate; // expirationDate for Packet Food // Fruit, Lunch meal, Breakfast, Dinner, //
    private boolean isVegetarian; // Packet food

    public Food(String n, int q, double p, String e, Boolean t) {
        super(n, q, p);
        expirationDate = e;
        isVegetarian = t;
    }

    public boolean isVegetarian() {
        return isVegetarian;
    }

    public String getExpirationDate() {
        return expirationDate;
    }

    public void addQuantity(int amount) {
        if (amount > 0) {
            setQuantity(getQuantity() + amount);
            System.out.println("Added " + amount + " " + getName() + ". Total quantity now: " + getQuantity());
        } else {
            System.out.println(x:"Invalid quantity. Please enter a positive number.");
        }
    }
}
```

Inheritance is a mechanism where one class is able to inherit attributes and methods from another class, this is due to the fact that inheritance works by creating a new class based on the existing class. To make it easier we will use the term Parent class as the Superclass and Child class as the Subclass. Because of the nature of this mechanism, both Parent and Child will have a “is-a” relationship where all the attributes and methods in Parent class will exist in Child class but not all Child class attributes and methods exist in Parent class.

We can see this mechanism in our code by referring to class “**Item**” and “**Food**” where Item is the Parent class and Food is the Child class. The child class, if you can see, has a keyword “**extends**” to show that it is inheriting all the attributes and methods in its parent class Item. This keyword is used to show to the compiler that the class is a specialised class and its attributes and methods are bigger than what is stated in the class code.

Inheritance mechanism uses the keyword “**super**” to invoke the method in its parent class such as “**super(n, q, p)**” that will call the method in the parent class and add it to the child class. The keyword “super” is rather sensitive as it needs to be as the first statement of the method and it must not be together with “**this**” as they will break the code.

4. POLYMORPHISM

```
abstract class Location {
    private String name;
    private String description;

    public Location(String name, String description) {
        this.name = name;
        this.description = description;
    }

    public Location() {
    }

    public String getName() {
        return name;
    }

    public String getDescription() {
        return description;
    }

    public abstract String displayLocationInfo();
}
```

```
class Campground extends Location {
    private int sitesAvailable;

    // Constructor
    public Campground(String name, String description, int sitesAvailable) {
        super(name, description);
        this.sitesAvailable = sitesAvailable;
    }

    public Campground() {
        super();
    };

    public int getSitesAvailable() {
        return sitesAvailable;
    }

    public String displayLocationInfo() {
        String str = String.format(format:"Campground Name: %s\nDescription: %s\nSites Available: %d", getName(),
        getDescription(), sitesAvailable);
        return str;
    }
}
```

Next is Polymorphism which is a feature in Java that enables us to use the same method call and have different outcomes depending on its class. This is beneficial to us when we want to Abstract or Interface methods in our class while making our code more simple and secure. The reason this was said, compare to Inheritance, it will still create a method and its statement while using an Overriding method as the method's name are all the same, polymorphism takes a different approach by not providing the methods statement in its Main class and said method will not show in the other classes UML diagram.

As an example we can see the class “**abstract class Location**” where the keyword “**abstract**” is used to invoke polymorphism in one of its methods. This can be seen with the method “**public abstract String displayLocationInfo():**” where it will end with a semicolon and not a curly-bracket. We can also see that class Location does not provide any description to the method as we will put it in the other classes.

To prove this, we can see in class Campground where we can refer to the method “**public String displayLocationInfo()**”. This method is undoubtedly from class Location as we are calling the functions of “**getName()**” and “**getDescription()**” in its statement. By doing this method, we are making a method signature without implementations.

5. EXCEPTION HANDLING

```
public class DateAndTime {
    private String date;
    private String time;

    public DateAndTime(String date, String time) {
        this.date = date;
        this.time = time;
    }

    public String getDate() {
        return date;
    }

    public String getTime() {
        return time;
    }

    // Method untuk check date format
    public static boolean isValidDate(String date) {
        SimpleDateFormat dateFormat = new SimpleDateFormat(pattern:"yyyy-MM-dd");
        dateFormat.setLenient(false);
        try {
            dateFormat.parse(date);
            return true;
        } catch (ParseException e) {
            return false;
        }
    }

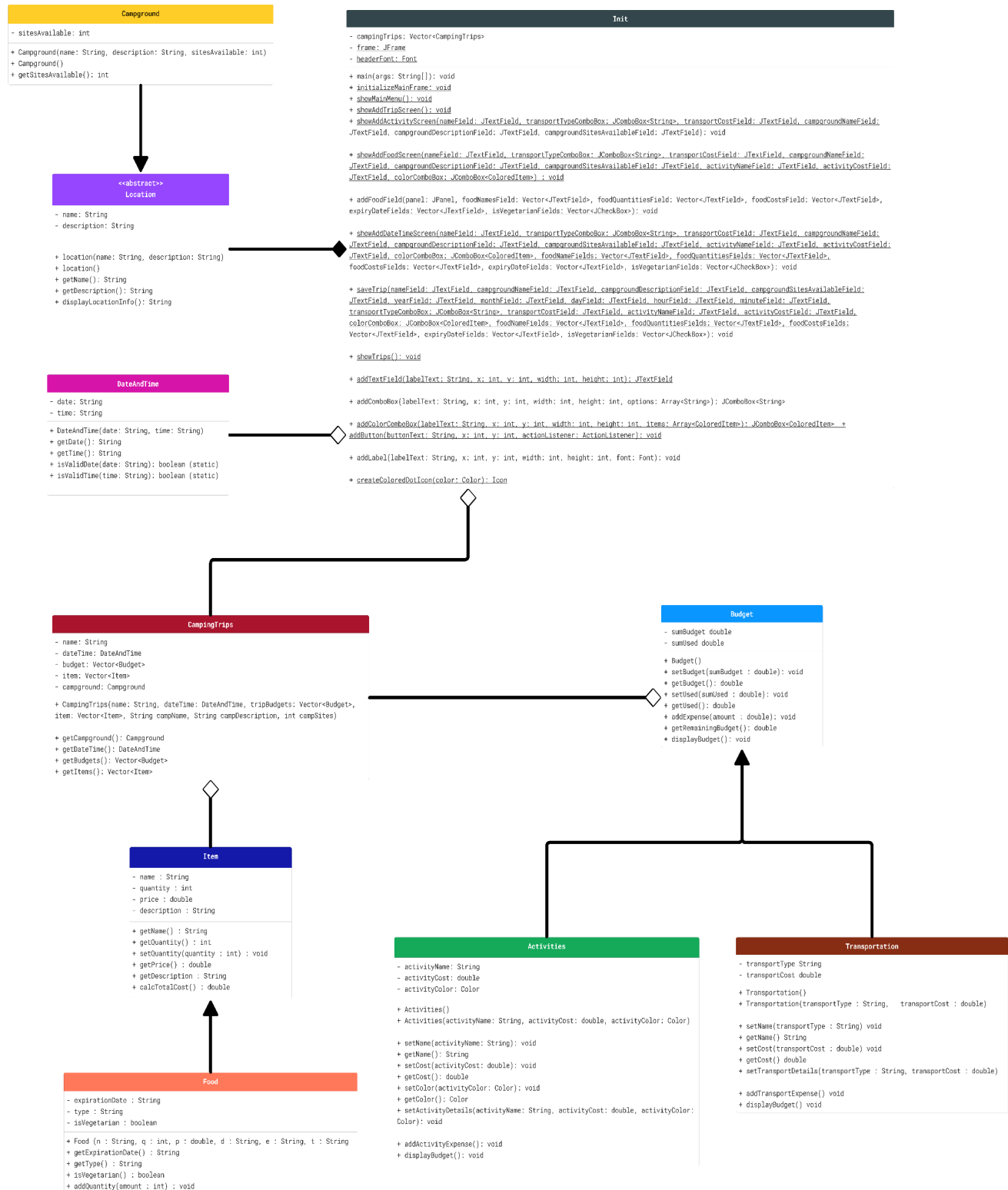
    // Method untuk check time format
    public static boolean isValidTime(String time) {
        SimpleDateFormat timeFormat = new SimpleDateFormat(pattern:"HH:mm");
        timeFormat.setLenient(false);
        try {
            timeFormat.parse(time);
            return true;
        } catch (ParseException e) {
            return false;
        }
    }
}
```

Exception handling is a runtime error event that will handle any errors that the compiler will give during execution time of a program. This is due to the fact that runtime errors are usually hard to debug as it comes from human errors such as wrong data type input, wrong file name or wrongly stated size of array. These are some of the errors that we humans will unfortunately make throughout our journey of coding and the way we can stop or find these problems are via implementing some exception handling in our code.

As we can see in our class `DateAndTime` in the method of “**public static boolean isValidDate**” and “**public static boolean isValidTime**” both have the keyword “**try**” and “**catch**” in its statements. These statements are the statements that will handle such errors if they do encounter it.

For an example is method “`public static boolean isValidDate`” will return a value of “**FALSE**” if it encounters a value that does not follow the date format of “`yyyy-MM-dd`” this will give an error notification to the user that they made a mistake in the Date and Time section in our code.

SECTION B: CLASS DIAGRAMS



Class Budget:

Attributes	Description
sumBudget	The total amount of Ringgit dedicated to the camping trip
sumUsed	The total amount of Ringgit used throughout the camping trip
Methods	Description
Budget	No-argument constructor
setBudget	To set the value of sumBudget
getBudget	To get the value of sumBudget
setUsed	To set the value of setUsed (for debugging)
getUsed	To get the value of setUsed
addExpense	Total the amount of Ringgit used in Transportation and Activities
getRemainingBudget	Calculate the remaining amount of sumBudget
displayBudget	Display window message to User to state their data has been inputted correctly

Class Transportation:

Attributes	Description
transportType	The type of transportation used
transportCost	The amount of budget needed for said transportation
Methods	Description
Transportation	No-argument constructor that refers to parent class Budget
Transportation(2-args)	To set the default value for transportType and transportCost and the values in parent class Budget
setName	To set the value of transportType (for debugging)
getName	To get the value of transportType
setCost	To set the value of transportCost (for debugging)
getCost	To get the value of transportCost
setTransportationDetails	Set the values of transportType and transportCost inputted by user into the attributes
addTransportExpense	Display window message to User to state their data has been inputted correctly and set the object value transportCost into the parent class Budget
displayBudget	Display window message to User to state their data has been inputted correctly while calling the method in parent class Budget

The work in this page has been done by : **DANIAL ERFAN SHAH BIN NOR AZAM SHAH**
Date:20/6/2024

Class Activities:

Attributes	Description
activityName	Stores the name of the activity.
activityCost	Stores the cost associated with the activity.
activityColor	Stores the color associated with the activity for visual representation.
Methods	Description
Activities()	No-argument constructor
Activities(3-args)	Constructor to set the initial values of activityName, activityCost, and activityColor.
setName(1-arg)	Sets the name of the activity.
getName()	Retrieves the name of the activity.
setCost(1-arg)	Sets the cost of the activity.
getCost()	Retrieves the cost of the activity.
setColor(1-arg)	Sets the color associated with the activity.
getColor()	Retrieves the color associated with the activity.
setActivityDetails(3-args)	Sets all details of the activity (name, cost, and color).
addActivityExpense()	Adds the activity cost to the overall budget and displays a message using JOptionPane.
displayBudget()	Overrides the displayBudget method from Budget class to display the activity name and cost using JOptionPane

Class DateAndTime:

Attributes	Description
date	Stores the date of the trip in the format "YYYY-MM-DD".
time	Stores the time of the trip in the format "HH:MM".
Methods	Description
DateAndTime(2-args)	Constructor to initialize date and time attributes.
getDate()	Retrieves the stored date of the trip.
getTime()	Retrieves the stored time of the trip.
isValidDate(String date)	Static method to validate if the provided date string matches "YYYY-MM-DD" format.
isValidTime(String time)	Static method to validate if the provided time string matches "HH:MM" format.

The work in this page has been done by : **NURAINN SOFEA BINTI NIZAMIL FAIRUZ**
Date:20/6/2024

Class Location:

Attributes	Description
name	Represents the name of the location
description	The details about the location.
Methods	Description
location()	Intializes the name and description attributes of the location.
getName()	Retrieves the name of the location.
getDescription()	Retrieves the description of the location.
displayLocationInfo()	Declares an abstract method that should be implemented by subclasses for specific information about the location.

Class Campground:

Attributes	Description
sitesAvailable	Number of available sites in the campground.
Methods	Description
campground()	Initializes the attributes of the campground.
getSitesAvailable()	Retrieves the number of available sites in the campground.

Class Item:

Attributes	Description
name	Stores the name of the item in String data type.
quantity	Stores the the quantity of the item
price	Stores the the price of the each item
description	Stores a string that contain details of the item
Methods	Description
getName()	Retrieves the data of the attribute name
getQuantity()	Retrieves the data of that stored in quantity
setQuantity(1-args)	Set the quantity of the item
getPrice()	Retrieves the data of the price of items
getDescription()	Retrieves the description of the item
calcTotalCost()	Calculate the total cost of the item by multiplying by the quantity

Class Food:

Attributes	Description
expirationDate	Stores the expiration date of a food.
type	Stores the type of the food.
isVegetarian	Stores a boolean value which shows vegetarian or not
Methods	Description
Food(6-args)	Constructor to initialise expirationDate,type,isVegetarian
getExpirationDate()	Retrieves the data of the expiration date of the food
getType()	Retrieves the data of the type of the food
isVegetarian()	Retrieves the boolean value of isVegetarian
addQuantity(1-args)	Add quantity of the food

Class Init:

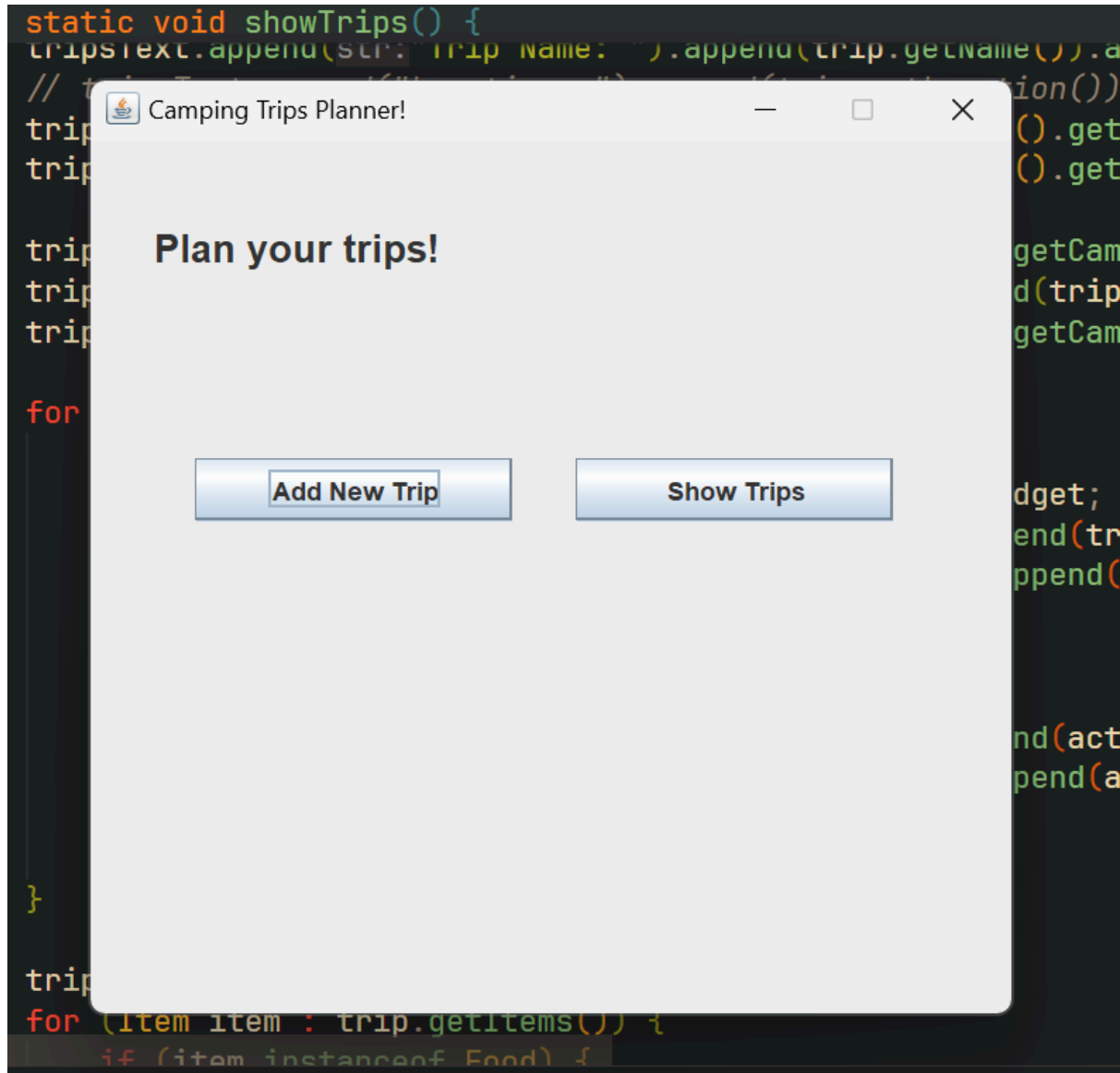
Attributes	Description
campingTrips	List of all camping trips that has been planned will be put inside this object (vector)
frame	Set a new frame screen with the title "Camping Trips Planner!"
headerFont	Set header label font to Arial, Bold, and size 18
Operations	Description
main	Where all the initialization user interface function will be called
initializeMainFrame	Set frame size width and height to 450. Set Default Close Operation to EXIT_ON_CLOSE Set Resizable to false
showMainMenu	Show the main screen, with button "Add New Trip" and "Show Trips", buttons when pressed will call their own screen
showAddTripScreen	Basically show the user interface all the trip information text field and label. There is also a button, when pressed will show to Activity screen
showAddActivityScreen	This screen shows all the Activity labels and text fields. There is a button when pressed will show the Food screen
showAddFoodScreen	This screen shows all the Food labels and text fields. There are two buttons "addFoodButton" and "addSaveButton". addFoodButton basically will call the addFoodField function, and addSaveButton will call the AddDateTimeScreen function.
addFoodField	Add more text field and label that is relates to food information
showAddDateTimeScreen	This screen shows all the Date&Time labels and text fields. There is a button when pressed will call the saveTrip function If the input is invalid it will show error message dialog
saveTrip	Will save all the trip into campingTrips vector, and then if the transportType is Miscellaneous, there will be an input message dialog that asks for the transport type.
showTrips	Basically loops through campingTrips vector and then all of the value inside it Apart from that there is also calculation process for all the budget, which is transport cost + activity cost + food cost
addTextField	Function that is used to create text field
addComboBox	Function that is used to create comboBox (selection)
addColorComboBox	Function that is used to create comboBox (selection but with colors)
addButton	Function that is used to add button
addLabel	Function that is used to add label
createColoredDotIcon	Function that is used to create color dot icon

Class CampingTrips:

Attributes	Description
name	Name of the camping trip
dateTime	Date and time of the camping trip
budget	List of budgets that has been made
item	List of the item that has been made
campground	Location information
Operations	Description
CampingTrips	Constructor that will grab the value and assign the value of name, dateTime, budget, item and campground
getCampground	Return the value of campground
getName	Return the value of name
getDateTime	Return the value of dateTime
getBudgets	Return the value of budget
getItems	Return the value of item

SECTION C: SOURCE CODE AND USER MANUAL

Main Screen



Initially, when the user runs our system from init.java they will be greeted with the home page, with a header label "Plan your trips!". Here they have 2 button options:-

- 1) **Add New Trip** - To create a new trip
- 2) **Show Trips** - To show previous trips that they have planned

Screen Add New Trip

The screenshot shows a mobile application window titled "Camping Trips Planner!". Inside the window is a form titled "Add New Trip". The form contains the following fields:

- Trip Name:** A text input field.
- Transport Type:** A dropdown menu with "Car" selected.
- Transport Cost:** A text input field.
- Campground Name:** A text input field.
- Campground Description:** A text input field.
- Sites Available:** A text input field.

A blue button labeled "Next" is positioned at the bottom right of the form.

First of all, in add New Trip Screen, there are several essential information that needs to be filled in by the user which is:-

- 1) **Trip Name** - Strings
- 2) **Transport Type** - This is selection field, not your ordinary text field
- 3) **Transport Cost** - Double
- 4) **Campground Name** - Strings
- 5) **Campground Description** - Strings
- 6) **Sites Available** - Integer

Explanation

Campground Name - Is basically where is the place that you plan to camp at

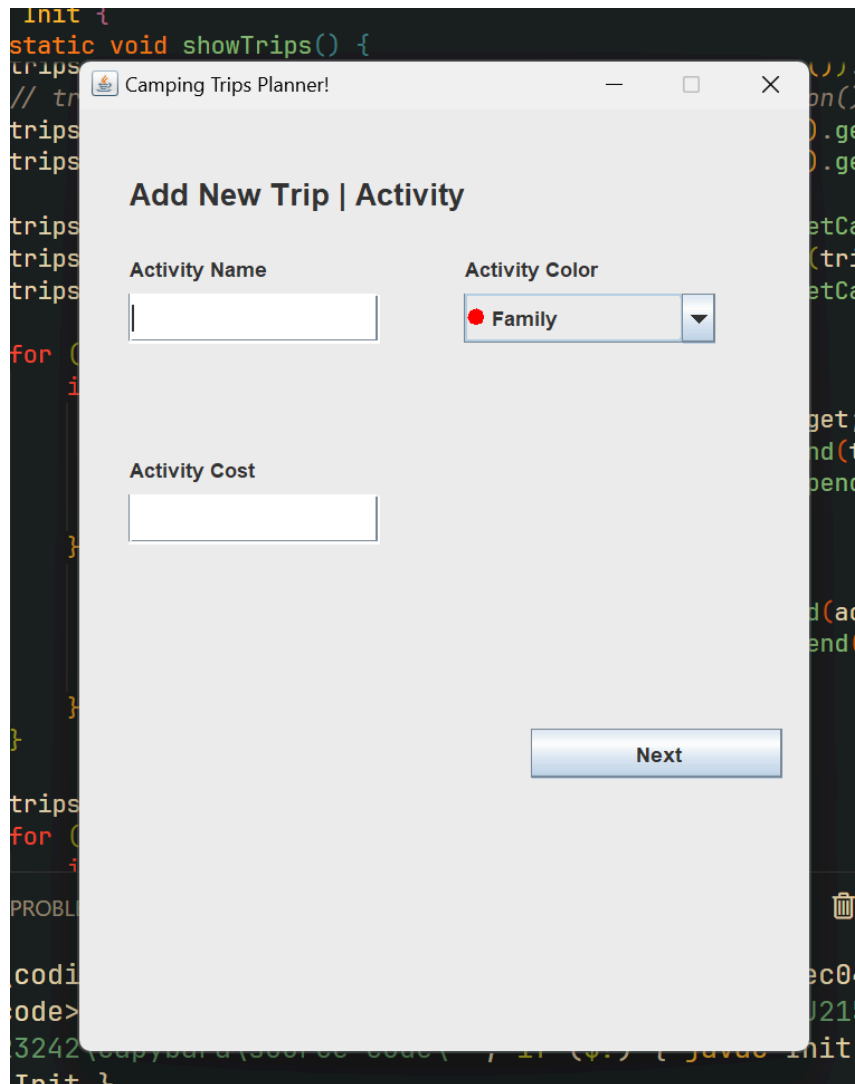
Campground Description - More precise location, eg. at the parking lot, besides the beach, and etc.

Sites Available - How many spots are available there?

After the user has already filled in all the information, they need to press the "Next" button and they will be greeted with another screen Activity.

Also, if the transport type selected is Miscellaneous, later on before saving the trip there will be an input dialog box asking for the user transport type in more detail.

Screen Activity



The screenshot shows a Java Swing window titled "Camping Trips Planner!". Inside the window is a form titled "Add New Trip | Activity". The form contains three input fields: "Activity Name" (a text box), "Activity Color" (a dropdown menu with "Family" selected and a red dot icon), and "Activity Cost" (a text box). A "Next" button is located at the bottom right of the form. The background of the image shows a code editor with Java code.

Here things that needs to be filled in are:-

- 1) **Activity Name** - Basically the activity that you are going to do with your groups
- 2) **Activity Type** - What type of activity it is, is it for family, friends, partner?
- 3) **Activity Cost** - How much that you need to spend for it

For Activity Type, we managed to again use selection field instead of text field, and each type has its own colour:-

- 1) **Red** - Family
- 2) **Green** - Friend
- 3) **Pink** - Partner
- 4) **Blue** - Others

After everything is done, pressing the “Next” button will bring to the food screen

Screen Food

The screenshot shows a window titled "Camping Trips Planner!". Inside, there's a section titled "Add New Trip | Food". It contains five text input fields labeled "Food Name", "Food Quantity", "Food Cost", "Expiry Date", and "Vegetarian" (which is a checkbox). Below these fields are two buttons: "Add More Food" and "Save Food". The "Add More Food" button is highlighted with a yellow border.

Before pressing Add More Food button

The screenshot shows the same "Add New Trip | Food" form, but now it has two rows of input fields. The first row is identical to the previous one. The second row has new input fields for "Food Name", "Food Quantity", "Food Cost", "Expiry Date", and "Vegetarian" (checkbox). The "Add More Food" button is still highlighted.

After pressing Add More Food button

This part right here is basically all of the informations regarding foods that you are planning to bring along, informations consist of:-

- 1) **Food Name** - Strings
- 2) **Food Quantity** - Integer
- 3) **Food Cost** - Double
- 4) **Expiry Date** - Strings
- 5) **Vegetarian** - Here is a type of field that is used is checkbox field

Apart from that, there are 2 buttons available, one is "Add More Food" the other one is "Save Food"

Add More Food will eventually add one more row of text field, into the screen so that the user can insert more than one food information

Save Food, will bring the user to the next page which is Date&Time screen

Screen Date&Time

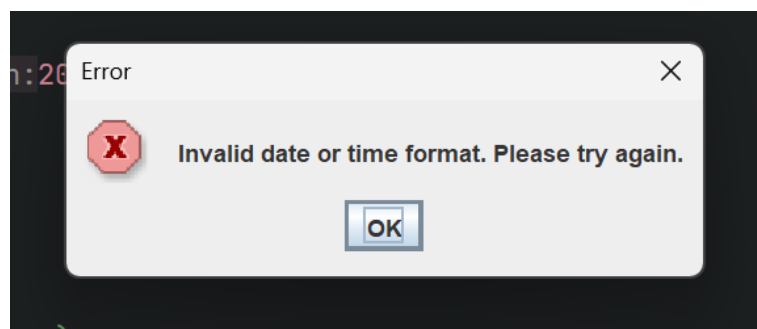
Camping Trips Planner!

Add New Trip | Date & Time

YYYY: MM: DD:

HH: MM:

Save Trip



This part is where the user puts when the camping event is going to start, therefore it consists of the date of the event, and also a specific timestamp of when the event is going to start.

YYYY - Year of the date

MM - Month of the date

DD - Day of the date

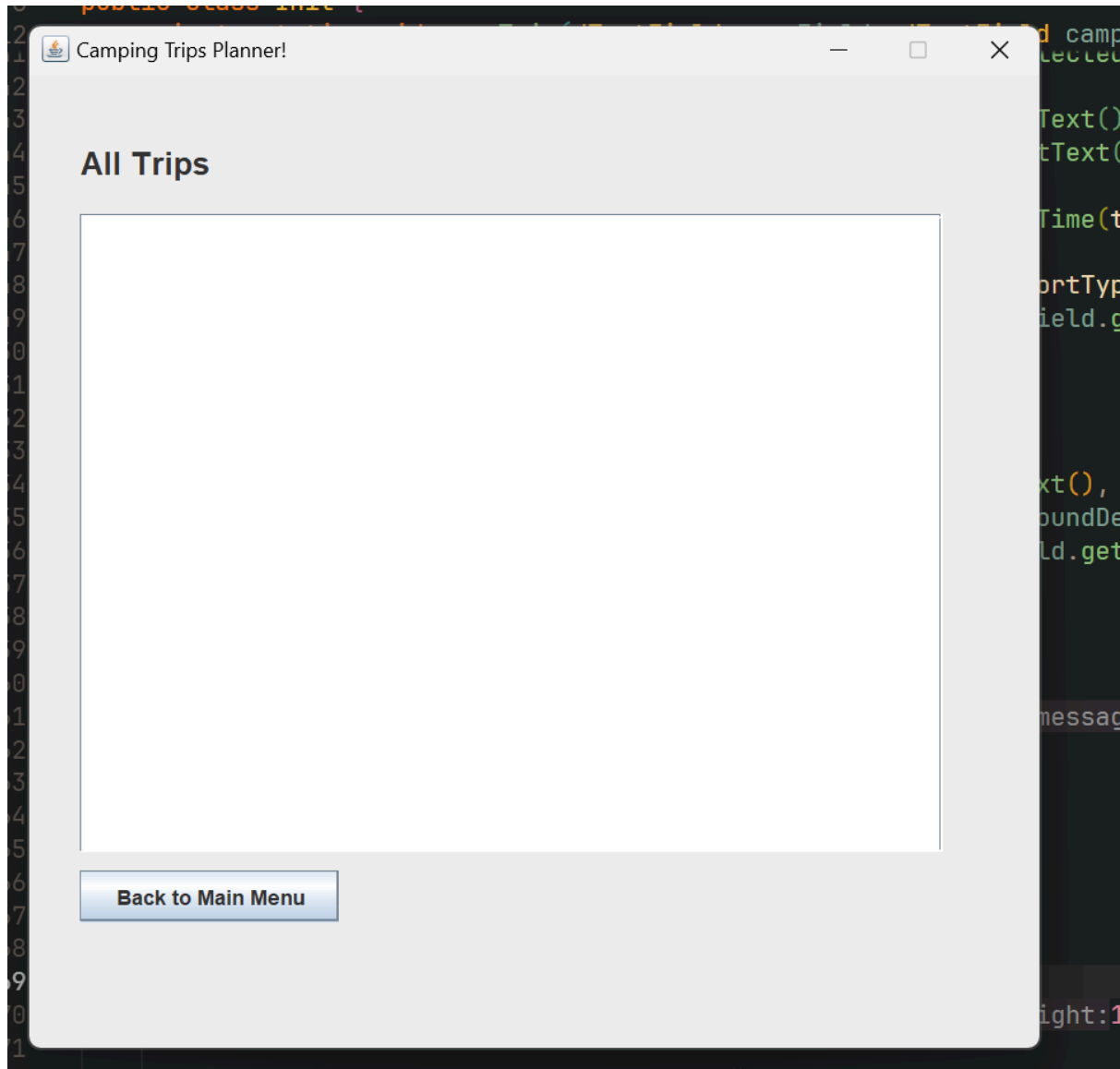
HH - Hour of the event

MM - Minutes of the event

Here if there is non logical input made by the user there will be an error message dialog indicating that the user needs to try again another suitable input instead.

When pressing the save trip button, it will bring you to the final output screen which is Show Trips screen

Screen Show Trips

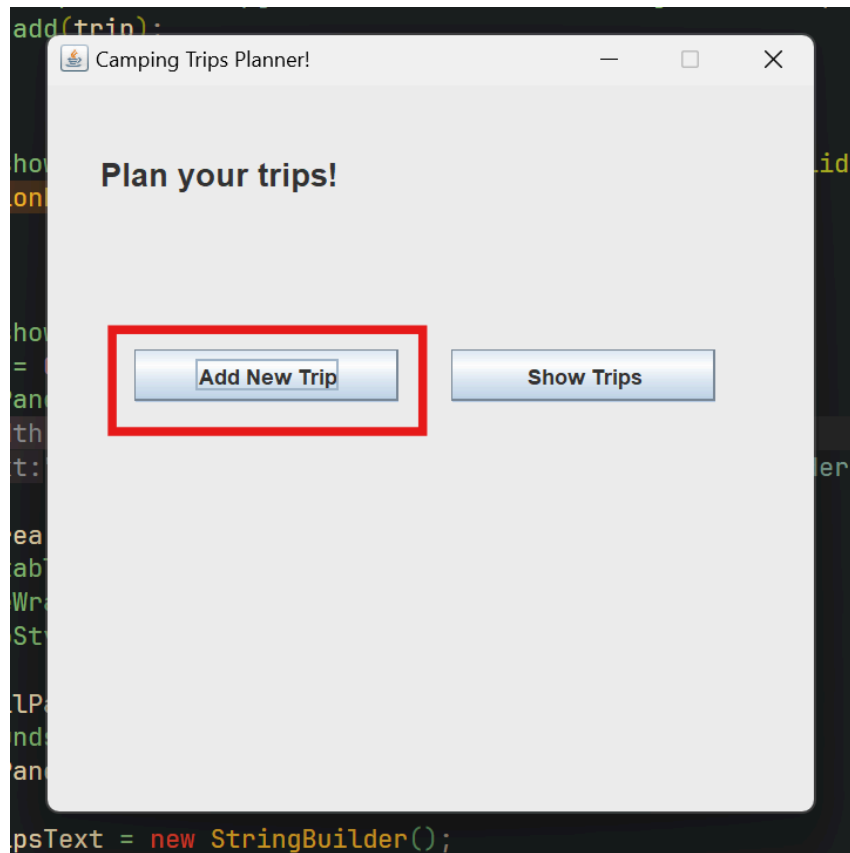


Here all the previous trips will be shown to the user. The most important feature that needs to be highlighted is, it **archives all trips**, the **total budget will also be automatically updated**. You can also **scroll the output box** too if the output is too long.

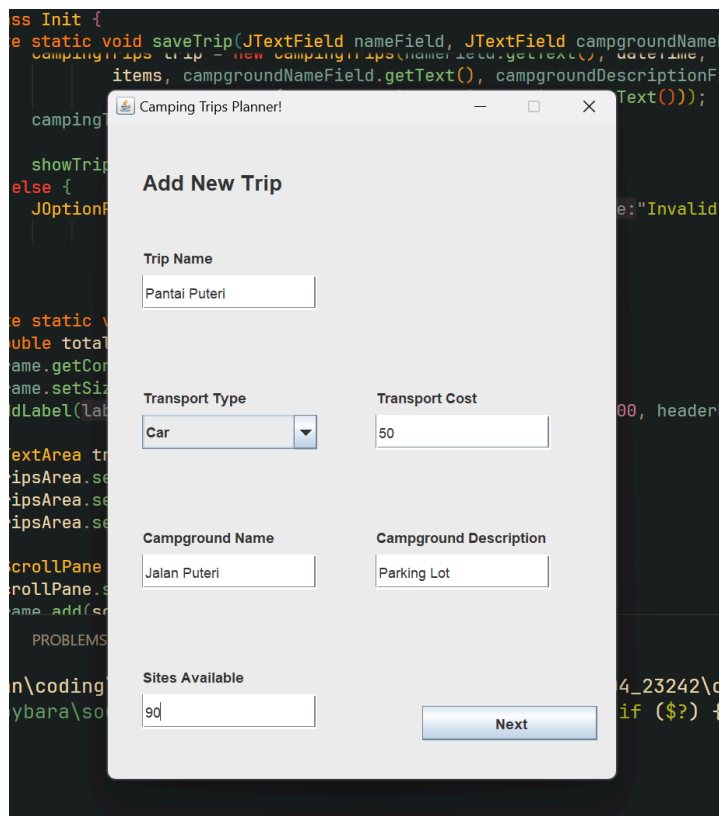
Pressing the "Back to Main Menu" button will bring it back to the Main screen.

Sample Input/Output

- 1) Press “Add New Trip” button



- 2) Fill in the trip information and press “Next” button



3) Fill in the activity information and press “Next” button

The screenshot shows a Java Swing window titled "Camping Trips Planner!". Inside, there is a dialog box titled "Add New Trip | Activity". The dialog contains three input fields: "Activity Name" with the text "Swimming", "Activity Color" with a dropdown menu showing "Friend" (indicated by a green dot), and "Activity Cost" with the value "4". A "Next" button is located at the bottom right of the dialog. The background shows a dark-themed IDE with Java code for a camping trip planner.

4) Fill in the food information, if there is more than one (in this case 2) press the “Add More Food” button and then press “Save Food” button to go to the next page

The screenshot shows the "Add New Trip | Food" dialog box. It features a table with two rows of food items. The first row is for "Ayam" with a quantity of 20, cost of 5, and expiry date of 12/12/2024. The second row is for "Roti" with a quantity of 10, cost of 1, and expiry date of 12/12/2024. Each row has a "Vegetarian" checkbox, which is unchecked for Ayam and checked for Roti. At the bottom, there are two buttons: "Add More Food" and "Save Food". The background shows the same IDE with Java code.

Food Name	Food Quantity	Food Cost	Expiry Date	Vegetarian
Ayam	20	5	12/12/2024	<input type="checkbox"/>
Roti	10	1	12/12/2024	<input checked="" type="checkbox"/>

- 5) Insert the camping trip event date & time and press “Save Trip” button to save the information



- 6) Lastly, the output screen will be shown, you can refer to your previous trips and also the total budget of the trip that you inserted. Pressing the “Back to Main Menu” button will bring you back to the Main screen.

