import java.io.\*;

import java.time.\*;

import java.time.format.\*;

import java.util.\*;

import java.util.concurrent.\*;

public class Main {

public static LocalDate currentDate = Calendars.GetCurrentDate();

public static void main(String[] args) throws IOException, InterruptedException {

Scanner input = new Scanner(System.in);

boolean run = true;

User.ReadUserFile();

//checks if program has been run before by calling CheckRunStartup, which returns false if no values exist in User file

if (User.CheckRunStartup()) {

System.out.println("Hello! Welcome to the neat calendar :D \n" +

"It is currently: " + Calendars.GetFormattedCurrentDate());

//runs startup method if program has not been run before

StartupMethod();

}

//prints the program opening title

System.out.println("<------------------------------------------------------------------------------------------------>\n" +

" 0000 0 0 00000 00 0 000 0 0000 |¯¯¯¯¯¯¯¯¯¯¯¯¯|\n" +

" 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |¯|¯|¯|¯|¯|¯|¯|\n"+

" 0 0 0 0 00000 0 0 0 0 0 0 0 0000 |-|-|-|-|-|-|-|\n" +

" 0 0 00000 0 0 0 00 0 0 00000 0 0 |\_|\_|\_|\_|\_|\_|\_|\n" +

" 0000 0 0 00000 00000 0 0 000 0 0 0 0 |\_|\_|\_|\_|\_|\_|\_|\n" +

"<------------------------------------------------------------------------------------------------>");

Calendars.PrintCalendar(); //prints calendar

TimeUnit.SECONDS.sleep(2); //stop for 2 seconds before continuing, allows for full calendar to be seen

//reads Assignments file and puts values into arrays within the Main class, to display assignments

Assignments.ReadAssignmentsFile();

ArrayList<String> listOfAssignmentSubjects = Assignments.GetAssignmentSubjects();

ArrayList<String> listOfAssignmentNames = Assignments.GetAssignmentNames();

ArrayList<String> listOfAssignmentDueDates = Assignments.GetAssignmentDueDates();

//reads Assignment Times file and puts values into arrays within the Main class, to display assignments

Assignments.ReadAssignmentTimesFile();

ArrayList<String> listOfAssignmentTimesSubjects = Assignments.GetAssignmentTimesSubject();

ArrayList<String> listOfAssignmentTimesNames = Assignments.GetAssignmentTimesName();

ArrayList<String> listOfAssignmentTimesHours = Assignments.GetAssignmentTimesHours();

ArrayList<String> listOfAssignmentTimesDueDates = Assignments.GetAssignmentTimesDueDates();

//reads User file and puts values into arrays within the Main class, to display events

LocalDate birthday = User.GetBirthday();

ArrayList<String> listOfYearlyEventNames = User.GetYearlyEventNames();

ArrayList<String> listOfYearlyEventDescriptions = User.GetYearlyEventDescriptions();

ArrayList<LocalDate> listOfYearlyEventDates = User.GetYearlyEventDates();

//int variable to count the number of assignments due on the current day

int assignmentsDueTodayCount = 0;

ArrayList<String> assignmentsDueToday = new ArrayList<>();

//traverses through assignment due dates and checks to see if the any are equal to today's date

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

if (listOfAssignmentDueDates.get(i).equals(String.valueOf(currentDate))) {

//adds assignments due today to assignmentsDueToday arraylist

assignmentsDueToday.add(" - " + listOfAssignmentSubjects.get(i) + ": " + listOfAssignmentNames.get(i));

assignmentsDueTodayCount++;

}

}

//int variable to count the number of assignments that have to be worked on the current day

int assignmentToWorkOnCount = 0;

ArrayList<String> assignmentsToWorkOnToday = new ArrayList<>();

//traverses through assignment times due dates and checks to see if the any are equal to today's date

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

if (listOfAssignmentTimesDueDates.get(i).equals(String.valueOf(currentDate))) {

//adds assignment and time it has to be worked on in minutes to assignmentsToWorkOnToday arraylist

assignmentsToWorkOnToday.add(" - " + listOfAssignmentTimesSubjects.get(i) + ": " +

listOfAssignmentTimesNames.get(i) + " for " +

(Double.parseDouble(listOfAssignmentTimesHours.get(i)) \* 60) + " minutes");

assignmentToWorkOnCount++;

}

}

System.out.println("Hello " + User.GetUsername() + ", today is the " + Calendars.GetFormattedCurrentDate() + "! :D");

//prints yearly events and details if they match today's date

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

if (listOfYearlyEventDates.get(i).getMonth().equals(currentDate.getMonth()) && listOfYearlyEventDates.get(i)

.getDayOfMonth() == currentDate.getDayOfMonth()) {

System.out.println("You have a yearly event! " + listOfYearlyEventNames.get(i) + ": " + listOfYearlyEventDescriptions.get(i));

}

}

//display if birthday is today

if (birthday.getMonth().equals(currentDate.getMonth()) && birthday.getDayOfMonth() == currentDate.getDayOfMonth()) {

System.out.println("HAPPY BIRTHDAY! I wish you a very happy birthday.");

}

//ensures grammar is correct if there is one or more than one assignments due

String assignmentsDuePlurality;

if (assignmentToWorkOnCount == 1) {

assignmentsDuePlurality = " assignment";

} else {

assignmentsDuePlurality = " assignments";

}

//prints the description for assignments that are due today

if (assignmentsDueToday.isEmpty()) {

System.out.println("You have no assignments due today! :)");

} else {

System.out.println("You have " + assignmentsDueTodayCount + assignmentsDuePlurality + " due today:");

for (String assignmentDueToday : assignmentsDueToday) {

System.out.println(assignmentDueToday);

}

}

//ensures grammar is correct if there is one or more than one assignments to work on

String assignmentTimesPlurality;

if (assignmentToWorkOnCount == 1) {

assignmentTimesPlurality = " assignment";

} else {

assignmentTimesPlurality = " assignments";

}

//prints the description for assignments that must be completed

if (assignmentsToWorkOnToday.isEmpty()) {

System.out.println("You have no assignments to work on today! :)");

} else {

System.out.println("You have " + assignmentToWorkOnCount + assignmentTimesPlurality + " to work on today:");

for (String assignmentTodayDetails : assignmentsToWorkOnToday) {

System.out.println(assignmentTodayDetails);

}

}

//variable that is used to determine the number of days assignments should be printed ahead of time

//this variable can be changed to user's desire, so it must be read from the User file and brought over from User class

int dayThreshold = Integer.parseInt(User.GetRunStartupUpcomingAssignments());

boolean upcomingAssignments = false;

//prints upcoming assignments that are due if the day minus the dayThreshold variable is before the current day,

//and if the original day is not equal to the current day (current day assignments will go through as they meet criteria)

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

if (LocalDate.parse(listOfAssignmentDueDates.get(i)).minusDays(dayThreshold).compareTo(currentDate.plusDays(1)) < 0

&& LocalDate.parse(listOfAssignmentDueDates.get(i)).compareTo((currentDate)) != 0) {

if (!upcomingAssignments) {

//prints the title for upcoming assignments only once if there are upcoming assignments

System.out.println("Upcoming assignments due:");

}

System.out.println(" - " + listOfAssignmentSubjects.get(i) + ": " + listOfAssignmentNames.get(i) + " due on "

+ listOfAssignmentDueDates.get(i));

upcomingAssignments = true;

}

}

//if there are no upcoming assignments

if (!upcomingAssignments) {

System.out.println("No upcoming assignments due.");

}

//resets the upcomingAssignments boolean for use in upcoming assignment to work on

upcomingAssignments = false;

//prints upcoming assignments to work on

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

if (LocalDate.parse(listOfAssignmentTimesDueDates.get(i)).minusDays(dayThreshold).compareTo(currentDate.plusDays(1)) < 0

&& LocalDate.parse(listOfAssignmentTimesDueDates.get(i)).compareTo((currentDate)) != 0) {

if (!upcomingAssignments) {

//prints the title for upcoming assignments only once if there are upcoming assignments

System.out.println("Upcoming assignments to work on:");

}

System.out.println(" - Working on " + listOfAssignmentTimesSubjects.get(i) + ": " +

listOfAssignmentTimesNames.get(i) + " for " + listOfAssignmentTimesHours.get(i) + " hours on "

+ listOfAssignmentTimesDueDates.get(i));

upcomingAssignments = true;

}

}

//if there are no upcoming assignments

if (!upcomingAssignments) {

System.out.println("No upcoming assignments to work on.");

}

//checks if there are any assignments that have gone past the due date since last time running

boolean assignmentPastDueDate = false;

//transverses the list of assignment due dates, and if any are before the current day, they are past the due date

for (String listOfAssignmentDueDate : listOfAssignmentDueDates) {

if (LocalDate.parse(listOfAssignmentDueDate).compareTo(currentDate) < 0) {

//set the variable for going past due dates to be true

assignmentPastDueDate = true;

}

}

//call the AssignmentsPastDueDate method to get user to mark whether assignments are done or not

if (assignmentPastDueDate) {

Assignments.AssignmentPastDueDate();

}

//prints calendar whenever you exit back to the menu, except upon opening the program (printed with other

//information so you don't want a double print, so don't do it only when initially opening program)

int printMenuCount = 0;

boolean printCalendarAfterInitialPrint = false;

while (run) {

if (printMenuCount >= 1) {

printCalendarAfterInitialPrint = true;

}

if (printCalendarAfterInitialPrint) {

Calendars.PrintCalendar();

}

printMenuCount++;

//prints the menu options

System.out.println("\nPlease enter what you would like to do:\n" +

"1. Create Assignments/Reminders 2. Delete Assignment/Delete Event 3. Games " +

"4. Create Yearly Events 5. Options\n" +

" < 6. Previous month 7. Display Day/Assignment Finished " +

"8. Next Month >");

boolean error = true; //boolean for error handling for do while loop

int choice;

exitDeleteOption:

do {

try { //try catch to catch errors involving error in input for main menu

choice = input.nextInt();

input.nextLine();

switch (choice) {

//create assignments or reminders

case 1 -> {

AssignmentCreationMethod();

error = false;

}

//delete assignments or yearly events

case 2 -> {

int deleteOption;

//opens up menu for further choices

System.out.println("Enter -1 at anytime to exit back to menu.\n" +

"What would you like to delete? 1. Assignments 2. Yearly Events");

//loop that only breaks when -1 entered or proper value entered, else repeat indefinitely

while (true) {

deleteOption = input.nextInt();

if (deleteOption == -1) { //-1 exit switch breaks external loop to go back to menu

System.out.println("Exiting to menu.");

break exitDeleteOption;

} else if (deleteOption == 1) { //entering 1 goes to the DeleteAssignment() method

Assignments.DeleteAssignment();

error = false; //no error in input

break; //breaks out of while loop

} else if (deleteOption == 2) { //entering 2 goes to the DeleteYearlyEvent() method

User.DeleteYearlyEvent();

error = false; //no error in input

break; //breaks out of while loop

} else { //input was not what was expected, ask user to enter valid value

System.out.println("Please enter a valid option or -1 to exit back to menu.");

}

}

}

//games

case 3 -> {

//prints all the games

System.out.println("Enter -1 at anytime to exit back to menu.\n" +

"What game would you like to play?\n" +

"1. Nim (No tokens needed) 2. Battleship (1 Token required)\n" +

" 3. Yahtzee (2 Tokens required, but not for practice)\n" +

" 4. Bowling (3 Tokens required, but not for practice)");

//loop that only breaks when -1 entered or proper value entered, else repeat indefinitely

while (true) {

int gameOption = input.nextInt();

if (gameOption == -1) {

System.out.println("Exiting to menu.");

break exitDeleteOption;

} else if (gameOption == 1) {

Games.Nim();

error = false;

break;

} else if (gameOption == 2) {

if (Games.GetGamePoints() == 0) {

System.out.println("You don't have enough points to play.");

break exitDeleteOption;

}

Games.Battleship();

error = false;

break;

} else if (gameOption == 3) {

Games.Yahtzee();

error = false;

break;

} else if (gameOption == 4) {

Games.Bowling();

error = false;

break;

} else {

System.out.println("Please enter a valid option or -1 to exit back to menu.");

}

}

}

//create yearly events

case 4 -> {

User.CreateYearlyEvent();

error = false;

}

//options

case 5 -> {

//prints all the options

System.out.println("Enter -1 at anytime to exit back to menu.\n" +

"Please choose an option:\n" +

"1. Display System Time 2. Change amount of days assignments are displayed in advance\n" +

"3. Change Username 4. Instructions/How to Use the Program 5. Display number of Game Tokens\n" +

" 6. Exit Program 7. Reset Program");

//loop that only breaks when -1 entered or proper value entered, else repeat indefinitely

while (true) {

int optionOption = input.nextInt();

input.nextLine();

if (optionOption == -1) {

System.out.println("Exiting to menu.");

break exitDeleteOption;

} else if (optionOption == 1) {

Calendars.DisplayExactTime();

error = false;

break;

} else if (optionOption == 2) {

User.ChangeDayThreshold();

error = false;

break;

} else if (optionOption == 3) {

error = false;

break;

} else if (optionOption == 4) {

PrintInstructions();

error = false;

break;

} else if (optionOption == 5) {

//directly prints number of game tokens

System.out.println("The number of tokens you have are " + Games.GetGamePoints());

error = false;

break;

} else if (optionOption == 6) {

//turns all external error handling loops false, breaks out of main menu try catch

System.out.println("Goodbye " + User.GetUsername() + "!!!");

run = false;

break exitDeleteOption;

} else if (optionOption == 7) {

System.out.println("Are you sure you would like to reset the program? (Enter YES to confirm)");

String resetConfirmation = input.nextLine();

//confirms user wants to restart the program

if (resetConfirmation.equals("YES")) {

//deletes all files, essentially deleting all saved information (terminates program)

User.DeleteUserFile();

Assignments.DeleteAssignmentsFiles();

Games.DeleteGamesFile();

System.out.println("Deleted all files. \n\nTerminating program.");

//turns all external error handling loops false, breaks out of main menu try catch (terminates program)

run = false;

break exitDeleteOption;

} else {

//else if confirmation word not entered, return to menu

error = false;

break;

}

//if improper value entered for options in options menu, prompt user to retry

} else {

System.out.println("Please enter a valid option or -1 to exit back to menu.");

}

}

}

//shift calendar to previous month

case 6 -> {

Calendars.PreviousMonth();

error = false;

}

//display a day or mark assignment as complete

case 7 -> {

//opens up menu for further choices

System.out.println("Enter -1 to exit back to menu\n" +

"What would you like to do? 1. Display Day 2. Mark Assignment as Completed");

int displayDayAssignmentCompleteOption;

//loop that only breaks when -1 entered or proper value entered, else repeat indefinitely

while (true) {

displayDayAssignmentCompleteOption = input.nextInt();

input.nextLine();

if (displayDayAssignmentCompleteOption == -1) {

System.out.println("Exiting back to menu.");

break exitDeleteOption;

} else if (displayDayAssignmentCompleteOption == 1) {

Assignments.DisplayDayMethod();

break;

} else if (displayDayAssignmentCompleteOption == 2) {

Assignments.AssignmentCompleteMethod();

break;

} else {

System.out.println("Please enter proper values or -1 to exit.");

}

}

error = false;

}

//shift calendar to next month

case 8 -> {

Calendars.NextMonth();

error = false;

}

default -> System.out.println("Please enter a valid option.");

}

//if error when entering value for main menu, prompt user to retry

} catch (Exception e) {

System.out.println("Please enter a valid option.");

input.next(); //consumes token so won't read the same value again

break;

}

//printing menu repeats as long as there is an error

} while (error);

}

}

//takes user input for a new assignment, then passes to Assignment class for the assignment to be written in

public static void AssignmentCreationMethod() {

Scanner input = new Scanner(System.in);

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MMMM d, yyyy", Locale.ENGLISH);

System.out.println("Enter -1 at anytime to exit back to the menu and terminate current process.");

System.out.println("Please enter the subject of your new assignment/reminder:\n(Please enter 'Social', " +

"'English', 'Math', 'Science', or 'Other')");

String subject;

while (true) {

subject = input.nextLine();

if (subject.equals("-1")) {

System.out.println("Exiting to menu.");

return;

} else if ((!subject.equals("Social")) && (!subject.equals("English")) && (!subject.equals("Math")) &&

(!subject.equals("Science")) && (!subject.equals("Other"))) {

System.out.println("Please ensure you enter proper formatting of the subject.");

} else {

break;

}

}

System.out.println("Please enter the name of the assignment/reminder:\n(Be descriptive)");

ArrayList<String> listOfAssignmentNames = Assignments.GetAssignmentNames();

String assignmentName;

boolean assignmentNameDuplicateCheck;

do {

assignmentNameDuplicateCheck = false;

assignmentName = input.nextLine();

for (String listOfAssignmentName : listOfAssignmentNames) {

if (assignmentName.equals(listOfAssignmentName)) {

System.out.println("An assignment already has that name. Please input another name or put numbers after.");

assignmentNameDuplicateCheck = true;

}

}

} while (assignmentNameDuplicateCheck);

String assignmentDueDate = null;

LocalDate formattedAssignmentDueDate = null;

boolean dateError = true;

System.out.println("Please enter the due date of the assignment/reminder:\n(In the format Month Day, Year)");

do {

try {

assignmentDueDate = input.nextLine();

//instantly tries to format the due date to see if it was inputted correctly, if not goes to catch

formattedAssignmentDueDate = LocalDate.parse(assignmentDueDate, formatter);

//compares due date entered with current date, does not allow due to date be before current day

if (formattedAssignmentDueDate.compareTo(currentDate) < 0) {

System.out.println("Assignment due date cannot be before today's date.");

} else if (assignmentDueDate.equals("-1")) {

System.out.println("Exiting to menu.");

return;

} else {

dateError = false;

}

} catch (Exception e) {

//asks user to try again

System.out.println("There was an error in your format. Please try again.");

input.next();

}

} while (dateError);

String assignmentCompletionDate;

LocalDate formattedAssignmentCompletionDate = null;

boolean dateError2 = true;

System.out.println("Please enter the date you would like to complete the assignment/reminder by:" +

"\n(In the format Month Day, Year (Enter 0 if this is a reminder or if you want to finish on the due date))");

do {

try {

assignmentCompletionDate = input.nextLine();

if (assignmentCompletionDate.equals("0")) {

assignmentCompletionDate = assignmentDueDate;

}

formattedAssignmentCompletionDate = LocalDate.parse(assignmentCompletionDate, formatter);

if (formattedAssignmentCompletionDate.isAfter(formattedAssignmentDueDate)) {

System.out.println("Assignment completion date cannot be after assignment due date. Please enter another value.");

} else if (assignmentCompletionDate.equals("-1")) {

System.out.println("Exiting to menu.");

return;

} else {

dateError2 = false;

}

} catch (Exception e) {

System.out.println("There was an error in your format. Please try again.");

}

} while (dateError2);

System.out.println("Please enter the amount of hours you estimate are needed to complete the assignment/reminder:" +

"\n(Enter 0 if this is a reminder)");

double assignmentHours;

while (true) {

assignmentHours = input.nextDouble();

if (assignmentHours == -1) {

System.out.println("Exiting to menu.");

return;

} else if (assignmentHours < 0) {

System.out.println("Hours required for assignment cannot be below 0.");

} else {

break;

}

}

int assignmentDays;

if (assignmentHours == 0) {

System.out.println("You are creating a reminder, so amount of days to complete the assignment is skipped.");

assignmentDays = 0;

} else {

System.out.println("Please enter the amount of days you believe is needed to complete the assignment:" +

"\n(Whole number digit, 0 if reminder)");

while (true) {

assignmentDays = input.nextInt();

if (formattedAssignmentCompletionDate.minusDays(assignmentDays - 1).compareTo(currentDate) < 0) {

System.out.println("With days inputted, starting date is before today's date, which is not possible. Please try again.");

} else if (assignmentDays == -1) {

System.out.println("Exiting to menu.");

return;

} else if (assignmentDays < 0) {

System.out.println("Assignment days cannot be below 0.");

} else {

break;

}

}

}

Assignments newAssignment = new Assignments(subject, assignmentName, formattedAssignmentDueDate, assignmentHours,

assignmentDays, formattedAssignmentCompletionDate);

newAssignment.AssignmentConfirmation();

}

public static void PrintInstructions() {

System.out.println("CALENDAR INSTRUCTIONS AND HOW TO USE:\n" +

"This calendar is tailored to your every school need, with a main visual interface for easy\n" +

"access to all of your assignments. Below is a detailed list of all the functions and what they do. All\n" +

"information inputted is stored in text files, to retain information between sessions and save the\n" +

"information you have inputted.\n" +

"Do note that this instruction manual is available to you anytime through the options submenu.\n" +

"1. Create Assignments/Reminders:\n" +

" This function allows you to create assignments or reminders, and allows the input for the amount\n" +

" of hours and days you estimate are needed to complete an assignment, at which point the program\n" +

" will perform the proper calculations and calculate the amount of time you need to work on and\n" +

" assignment everyday leading up to the due date.\n" +

"2. Delete Assignment/Delete Event:\n" +

" This function allows you to delete an assignment or event with the input of the name of the\n" +

" assignment or event. This opens up the possibility of removing any information that has been\n" +

" inputted incorrectly or is no longer needed.\n" +

"3. Games:\n" +

" This program comes with 3 games; Nim, Battleship, Yahtzee, and Bowling. Nim is free to play for as\n" +

" many times as you want, Battleship requires tokens to play that you get when you have completed an\n" +

" assignment and marked it as completed. Yahtzee and Bowling are small programs by themselves, and\n" +

" allow for free infinite practices that consist of one practice round, but to play a full game\n" +

" requires tokens. The scores for all games are stored by the program (win ratio or high score).\n" +

"4. Create Yearly Events:\n" +

" Allows for the creation of yearly events, which as the name says occurs every year, which is\n" +

" displayed on the visual calendar. Additionally, you can write your own descriptions that will\n" +

" display when it is the day of the event.\n" +

"5. Options:\n" +

" The options function has 6 choices inside, all of which allow you to edit aspects of the program.\n" +

" 1. Display System Time: Displays the exact time used in the program, allowing verification by user\n" +

" 2. Change amount of days assignments are displayed in advance: Within this program, assignments\n" +

" can be displayed in advance, this option allows you to specify the number of days assignments\n" +

" are displayed in advance which is displayed upon opening the program. The default value is 1.\n" +

" 3. Change Username: If you do not like your current username, you are allowed to change it an\n" +

" infinite number of times\n" +

" 4. Instructions/How to Use the Program: Displays the instructions menu you are reading right now\n" +

" 5. Display Number of Game Tokens: Displays the exact amount of tokens you currently have\n" +

" 6. Exit Program: Allows you to exit the program and terminate all processes if necessary\n" +

" 7. Reset Program: Self explanatory. Erases all text files and resets the program, requiring\n" +

" confirmation first\n" +

"6. Previous Month:\n" +

" Moves the calendar to the previous month, displayed on the visual calendar (with all assignments and events)\n" +

"7. Display Day/Assignment Finished:\n" +

" Display day allows you to input a specific date, and the program will display all assignments that\n" +

" are due on that day, must be worked on that day, and events that are happening on that day.\n" +

" Assignment finished allows the user to tell the program they have finished an assignment, and\n" +

" rewards the user with game points for having finished an assignment.\n" +

"8. Next Month:\n" +

" Moves the calendar to the next month, displayed on the visual calendar (with all assignments and events)\n");

}

//prompts user to enter their username and birthday then passes information to user class to be processed

public static void StartupMethod() {

Scanner input = new Scanner(System.in);

//formatter variable to format the day the user enters

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MMMM d, yyyy", Locale.ENGLISH);

System.out.println("Please enter your username:");

String username = input.nextLine();

System.out.println("Hello " + username + " nice to meet you! :)");

System.out.println("Please enter the year of your birthday (in the format Month Day, Year):");

String birthday = input.nextLine();

try {

//formats the birthday the user enters and parses it from a String to a Localdate variable

LocalDate formattedBirthday = LocalDate.parse(birthday, formatter);

User newUser = new User(username, formattedBirthday); //creates a new user object, information is written to User file

newUser.CalculateAge(formattedBirthday); //calculates the users age

System.out.println("Welcome to your new calendar " + User.GetUsername() + "! :)\n");

TimeUnit.SECONDS.sleep(3); //gives user time to read all the outputs before printing the instructions

//prints the instructions

PrintInstructions();

System.out.println("\nPlease read the instructions, then enter any key to continue and enter the calendar.");

input.nextLine(); //waits until next user input to continue code, gives time to read instructions

} catch (Exception e) { //asks user to enter values again if error comes up

System.out.println("Please ensure you are entering values in the proper format.");

StartupMethod(); //recursive method call, repeats asking if error occurs

}

}

}