Software Documentation

Course: [COP-2805C-86399 Java Advanced](https://hcc.instructure.com/courses/135902)

Assignment: Demonstrate OOP principles for Inheritance

Date: 2/18/24 (Edited 2/24/24)

Author Details: Jose A. Cañellas

# Requirements

* The program must include three classes, Event, BirthdayParty that extends Event, and Quinceanera that extends BirthdayParty. It must also include one called EventPlanner with a runnable Main method that will request the user the number of events they wish to make, assign an event type to each object, and fill in an Event array with the objects of the Classes chosen.
* The Event Class must have the following:
  + Setter and Getter methods for:
    - The date of the event.
    - The name of the event.
    - The start and end time of the event.
    - The location name of the event.
    - The address of the event’s location.
    - The number of guests attending .
    - The point of contact for the event.
    - The price of entry for the event.
  + A toString method to display the details for each Event object.
* The BirthdayParty Class must inherit from Event and have the following:
  + Setter and Getter methods for:
    - The celebrants name.
    - The age being celebrated.
    - The type of Cake.
    - The number of candles for the cake.
    - The theme for the decorations.
  + An overridden toString method to display the details of the BirthdayParty object.
* The Quinceanera Class must inherit from BirthdayParty and have the following:
  + Setter and Getter methods for:
    - The Quinceañera’s name.
    - The number of Damas.
    - The number of toasts performed.
    - The choice of dance music.
* The EventPlanner Class must have a runnable Main method and the following:
  + A prompt to ask the user for the number of events they wish to make.
  + An array of type Event of the same size as the user’s request.
  + A prompt to ask the user the type of events they wish to create so that the appropriate Class Objects are created and added to the Event array.
  + After setting the details for each Class Object, EventPlanner must display the details of each Object created using their respective toString methods.

# Installation and Run Instructions

* The user must have any Operating System compatible with the latest version of Visual Studio Code (Windows 10 and 11, Linux, MacOS 10.15 or above)
* The user must install Visual Studio Code and install the “Coding Pack for Java” in the following link (https://code.visualstudio.com/docs/languages/java) . The “Extension Pack” is also recommended.
* Four .java source files will be provided, and the user must open a Java project where they can drag these files into the “src” tab in Explorer.
* On the upper left, click run and then “Start Debugging” or “Run without Debugging” to run EventPlanner.java.

# Design Notes and UML Diagram

* When designing the program, the first step was to create the Event Class since everything else will come from there. The necessary variables were created, and the setter and getter methods were made without anything in them, that way there was a sort of template to start working from.
* When thinking about the different variables that needed to be set, the thought of how to handle the inputs from the user was constantly in the mind. Therefore, as the setter methods for each variable were made some sort of input verification was needed.
  + In the case of the dates and start/end times, the Java API was consulted, most importantly java.time as this handled most things to do with time and dates. For the date of the event, a lot of difficulty was encountered as many options were attempted but none of them worked the way that was wanted. I wanted the date to be in a specific format, and that if the format entered was incorrect then an error would be displayed. The issue was finding a way for the program to recognize if a number above 31 was entered on the day or a number above 12 was entered as the month. Chapter 10 of the book described the use of regular expressions (or regex) which is further explored in Appendix H of the book and while looking at the many examples of matching user input with various patterns for things like social security and phone numbers I thought this could be perfect for the date.

So, the idea was for the user input of the date to be passed onto a method called validInputDate and in it match it with a String called regex that contained the pattern format. The format in the string made sure that no number below 1 or above 31 was entered, no number under 1 or above 12 was entered for the month, and that no year below 2020 or above 2099 was entered. This was probably one of the most difficult parts of the project.

* + When it came to the time, I came upon LocalTime which can be used to store hours and minutes inside. Using LocalTime’s parse method, one can get the input from the user, and it will return the parsed time, but it throws a DateTimeParseException if the text can’t be parsed. Because of this exception, a Try/Catch block was made so that when the exception is thrown an error explaining to the user that their input was invalid is shown, then tries again to parse the newer input.
  + After parsing through the input, both setter methods for start time and end time extract the hours and minutes fields from the parsed input using LocalTime’s getHour and getMinute methods which return the extracted hours or minutes.
  + Then, the user is asked at which time the event starts or ends, AM or PM. Depending on their answer a String.format method is used by passing the hours and minutes extracted by parse() and using the format “%02d:%02d (AM or PM)” a String is created with the time and AM or PM is added at the end of it.
  + The next thing to deal with was an issue I encountered when working on Project 2, my main method was filled with repetitive text asking the user to fill out the different attributes of the cars or drivers. I thought hard on how to fix that but unfortunately couldn’t find the time to do so and resubmit, so I decided to try here. I tested the setter methods to see if they would prompt the user for the input and they did, so every setter method had the appropriate prompt for every variable. This would remove all those repetitive prompts from EventPlanner. It turned out to be much easier than thought so I will be using this way from now on.
    - I also made it so that when the prompt asks the user for an input, it would concatenate the name of the event given by the user to the prompt. Not really necessary but it looks better that way.
  + The next issue was trying to give the user a way to change their input if they wanted to make any corrections. Originally, most setter methods in Event had some way to do this but this just created more clutter and repetition in the code so the idea to put this in a method came to mind. A method named inputValidation was made to contain what most methods originally had, a prompt that would request a yes or no answer after asking if they wish to make a correction and depending on their input a correction was made, or the program would continue to the next setter. The problem was that the first variation of it only allowed one correction and quickly returned that first correction. For this, another loop was placed inside where the user is asked if their new input was correct, or they wished to make another correction. If they wish to do so they may make as many corrections as they wish and then return the corrected input which is then assigned to the appropriate variable. For now, this can only be done with Strings, but hopefully I may be able to make a method for Integers (**Update**: integer Input Validation successfully implemented).
* With the input validation dealt with, a method was added at the end of Event to run all setter methods one after the other called allSetMethods. This saved time and made the code for EventPlanner simpler as only one line was required for this. And finally, a toString method was created to display all the details of the object.
* For BirthdayParty, the only setter methods that had to be made were the ones for the variables specific to it which was very simple as Event contained all of the heavy lifting. The inputCorrection method in Event was applied to most setter methods since it was inherited from Event.
* Only two methods had to be changed, allSetMethods and toString. These two were overridden in both BirthdayParty and Quinceanera so the necessary details were set and then displayed correctly.
* For Quinceanera, the same thing was done with BirthdayParty, the necessary variables were set using their setter methods, some methods called the inputCorrection method from Event, and both allSetMethods and toString were overridden for the same purpose as BirthdayParty. The only big difference is that when the Constructor method is used, the age is set to 15 and allSetMethods doesn’t use the setBirthdayAge method from BirthdayParty.
* Finally, with all the Classes done, EventPlanner had to take shape.
  + First, it had to prompt the user for the number of events they wish to create. The input was then assigned to a variable called numberOfEvents to be used in the creation of the array and in a while loop so that the loop runs as long as a counter variable is lower than it since arrays start at 0.
  + Second, it had to prompt the user to pick the type of event each one would be, so that the appropriate Constructors were used. The first step was verifying that their input was an integer and for that scanner.hasNextInt was used in an if statement, if true it would proceed with another if statement to make sure the input is between 1 and 3. If withing the range, it proceeds with a switch statement with 3 cases, one for each option (1. Event, 2. Birthday, 3. Quinceanera). If the input is not an integer the appropriate error will be displayed and input requested once more, same thing for if the input was not in the range of 1-3.
  + Next, a while loop is used to call the allSetMethods of the appropriate class object for each one in the array and the details are filled. It keeps going as long as the counter is below the number of events. When that is done, the details of all the events created are then displayed.

**UML Diagram:**

A black screen with white text

Description automatically generated

A black screen with white text

Description automatically generated

# Test Plan

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Requirement | Input/Action | Expected | Output | Pass/Fail |
| All setter and getter methods set and get their variables | Used allSetMethods with example Class Objects. | Details for each Object are set correctly. | The toString method for each Class was used and the details were displayed correctly. | PASS |
| Input validation and correction methods work as designed | When assigning values to the different variables in all Classes, Incorrect inputs were entered on purpose. | The methods would catch any incorrect inputs, displaying their errors and allowing for new input or the option for correction was given. | The methods caught the incorrect inputs and the correction method allowed for corrections until the user was satisfied. | PASS |
| The parse method for times and the regex for dates work as designed | When input for those variables was requested, incorrect inputs were entered on purpose, then correct ones. | User would be prompted to enter the time again as format was incorrect, same thing for the date. | The incorrect inputs were caught, and the appropriate errors were displayed. When correct input was placed the variables were displayed correctly. | PASS |
| When requesting details for all Class Objects, the name of the event will be displayed when prompting for input. | When filling the details for every Object, a random name was assigned to each event. | The program prompts if the user wishes to correct their entry, if not, when other prompts appear the name of the event is included in the prompt. | The names assigned were correctly concatenated in each setter prompt. | Pass |
| When prompted for the number of events, the correct amount would be created | When prompted, asked to create 4, then 10, then 8. | An array with the requested amounts were created. | The array was created with the requested amounts and the correct objects were made when prompted. | PASS |

# Screenshots

A black screen with white text

Description automatically generated

A screenshot of a computer

Description automatically generated











