COP-2805C-03597

Java Advanced Programming

P2 - Inheritance

Document Version: 0.1

Version Date: 06/06/2025

Created by: Joseph Simonin

Version Control

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Editor | Comments |
| .01 | 06/06/25 | Joseph Simonin | 1st draft |
|  |  |  |  |
|  |  |  |  |

Technical Specifications

Link to source code through GitHub

AI Statement

For this program the assistance of AI was used as a learning assistant to deepen my understanding of constructor chaining and help with debugging some compiler errors. I asked targeted questions when encountering difficult areas in the assignment, such as how to properly structure constructors in an inheritance chain, and why certain logic in my scanner input was failing. The AI helped to clarify these topics with examples and explanations. I remained in control of building the classes, designing the inputs, writing the conditions for user interaction, and debugging through most of the errors I encountered. The AI’s role was like that of a tutor. It gave me advice when I asked for it and showed how to fix problems in a way that I could learn from and apply myself during future code. I estimate about 25% of this program was supported by the AI’s explanations and debugging suggestions. The remaining 75% of the work was my own including coding design choices, logic creation, testing, and final revisions. Overall, the AI served as a learning assistant that helped me improve my understanding and efficiency of the concepts covered this week.

Description of Requirement

Write a Java program with four classes in four separate source code files.

The first class is Event. Events should at a minimum have a date, start and end time, location name, location address, event name, number of guests, point of contact, and price. You may add additional attributes. Event should have a public toString() method.

The second class is BirthdayParty. A BirthdayParty is-a Event, but should also include appropriate attributes and methods for age, cake, candles, and decorations. BirthdayParty should override the parent's toString() method.

The third class is Quinceañera (you may replace the ñ with an English n for purposes of the program). A Quinceañera is a specific type of BirthdayParty. The age should be immutably set to 15. Additional attributes will include number of maids of honor, number of toasts performed, and choice of dance music. Quinceañera should override the parent's toString() method.

A Quinceañera is-a BirthdayParty.  A BirthdayParty is-a Event.

The fourth class is EventPlanner. EventPlanner will include a runnable main() method.

When run, the program should print out the course, assignment, current date, and student name to the screen. Then prompt the user for the number of events to create.  Then create an array of type Event sufficient to hold the number of events given by the user.

For each item in the array, prompt the user for the event type, choosing from the three types. Accept the user’s responses and create an object of that type and assign it to the array.

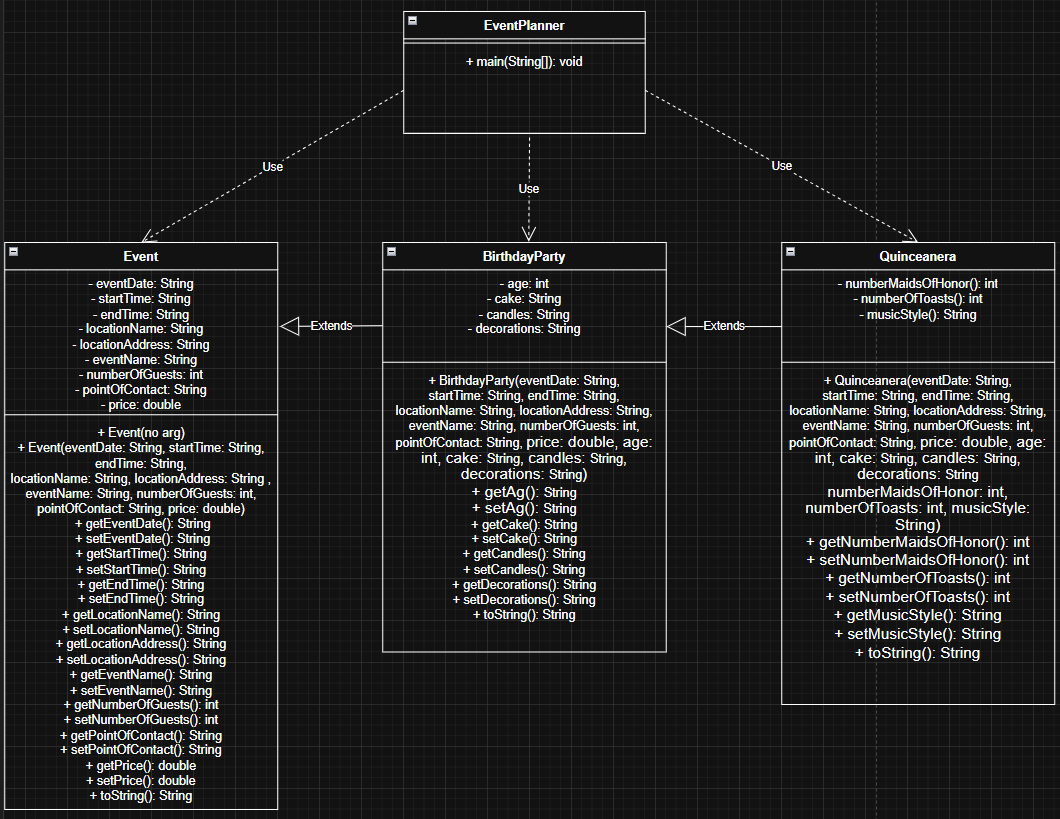
After all user-supplied objects are created, loop through the array and print out the event planning report showing the state of each event.

All classes should demonstrate inheritance, encapsulation, have appropriate get and set methods, and have a toString() method. Child classes should call parent constructors.

Installation and Run Instructions

Open all 4 .Java files into IDE. Select EventPlanner.java file and run the program. The program will prompt the user to enter a number for how many desired events to create. After taking user input for desired events to create the user will be prompted with three event types to select from(Event, Birthday Party, Quinceanera). Upon making the desired event type selection the user will be prompted with questions specific to each event type. The user will need to enter answers to questions. After all questions for all the events desired have been answered a final Event Report will be printed to the screen for the user to review.

UML Class Diagram



I will also attached a link to the chart In the submitted files incase this one is not readable from the screenshot

Design Notes

When designing the EventPlanner program I chose to use a for loop to collect user input for each event because the number of events was fixed and known ahead of time. A for loop provided a clear and structured way to loop through the array of events using indexing which was useful for assigning and accessing objects. I used inheritance to build the class structure. Events as the base class, BirthdayParty as a subclass of Event, and Quinceanera as a subclass of BirthdayParty. This allowed for reusable organized code while maintaining unique attributes for each type. A challenge I faced was understanding how to use constructor chaining to pass parameters from child classes to parent classes using super(), and making sure the arguments were in the correct order to avoid compiler errors. I learned how super.toString() Let’s child classes build upon the parent’s output. Another interesting part was using encapsulation to keep variables private and accessing them through getter and setter methods. For the Quinceanera class I learned how to enforce immutability by hard coating that age as 15 and omitting a setter for it. When working with user input, I discovered the importance of consistent input handling, especially reading different data types in sequence. Lastly, using a for-each loop to print the event planning report helped me see Java’s polymorphism allowing the correct version of toString() to be called based on an object’s actual type.

Program Validation

To validate that the Event Planner program met all project requirements, I conducted a series of functional tests involving different event types and varying user inputs. I began by entering "3" when prompted for the number of events to ensure the program correctly initialized an array of that size. For the first event, I selected type 1 (Event) and input typical event details such as a date, time, location, and price. I expected the resulting output to include all entered details formatted through the toString() method, which it did. For the second test, I selected type 2 (BirthdayParty) and added specific values for age, cake, candles, and decorations. The expected result was a printout combining both base Event data and additional BirthdayParty attributes. This also passed. For the final test, I selected type 3 (Quinceanera) and verified that age was automatically set to 15 regardless of user input. I entered values for dance music, toasts, and maids of honor and expected the report to show age as 15, which confirmed the immutability requirement. I further validated that all subclasses’ toString() methods correctly invoked super.toString() and appended their additional data. Each object printed as expected, demonstrating successful inheritance, encapsulation, and polymorphism. No runtime errors occurred, all user input prompts functioned correctly, and the final report loop correctly displayed the information for all events using a for-each loop. These results confirmed that the program fully met the specification.

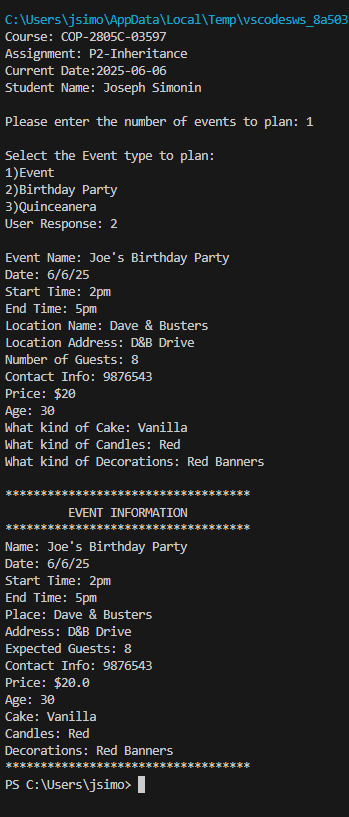
Screenshots of Tests

Test #1

A screen shot of a computer

AI-generated content may be incorrect.

Test #2



Test #3

A screen shot of a computer

AI-generated content may be incorrect.

Test #4

A screen shot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.