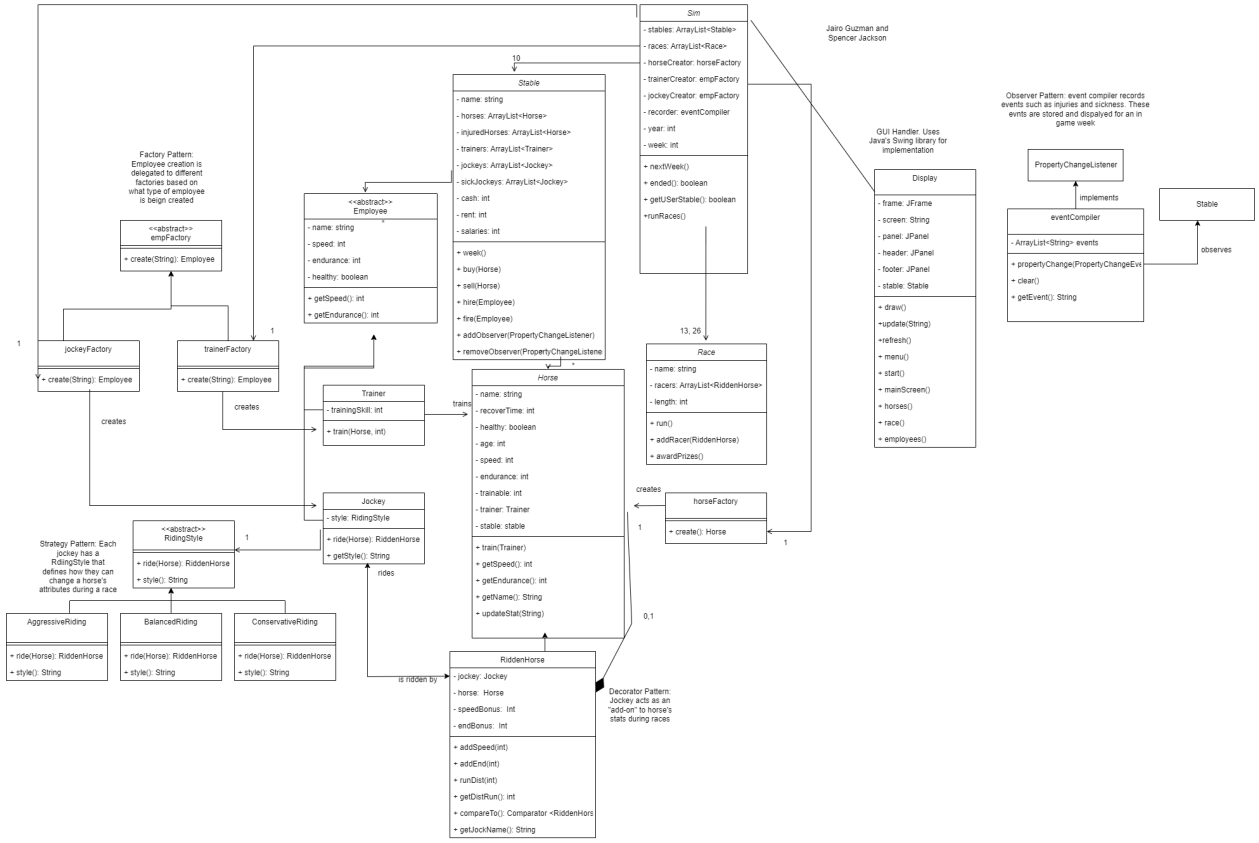
Project Summary

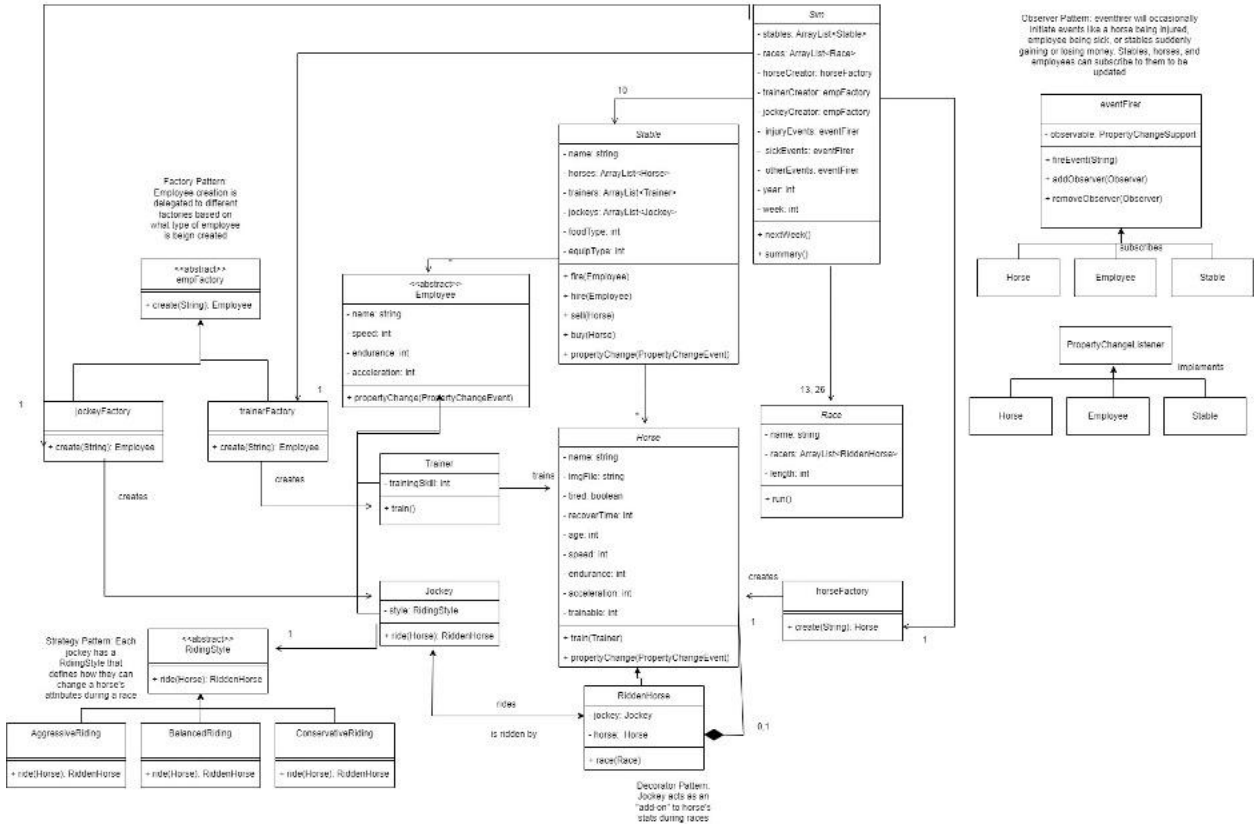
- Title: No Horsin’ Around
- Team Members: Spencer Jackson and Jairo Guzman
- The features included in our project is a main menu with other tabs to check for upcoming races, list of horses, list of employees and a tab to proceed into the next week. Names of horses and employees can be edited by the user and stats are given to each horse and employee. Compared to our original design only 2 out of 3 attributes were used in the final status of our project. The horses' stats will be able to change on a weekly basis as they will be trained by the trainers. The horses’ stats will also be changed in a race when matched with a jockey as each jockey has their own stats and riding style. The user will be presented with the list of horses and jockeys prior to the race. There is also news being shown which shows certain events such as a horse being injured or an employee getting sick. The horses and jockeys will be unavailable for races when these events occur. The user will also be able to fire employees and hire new employees. Horses can also be sold and new horses can be bought. In the main menu there is also a feature for how much money a user has as they win races or just expenses being paid. If the money ever reaches zero then the game is over as the user goes bankrupt. A feature we did not implement was to be able to save and load the game as we ran out of time.

Final Class Diagram and Comparison Statement

Final Diagram


Sim
- stables: ArrayList<Stable>
- races: ArrayList<Race>
- horseCreator: horseFactory
- trainerCreator: empFactory
- jockeyCreator: empFactory
- recorder: eventCompiler
- year: int
- week: int
+ nextWeek()
+ ended(): boolean
+ getUserStable(): boolean
+runRaces()
Jairo Guzman and Spencer Jackson
Stable
- name: string
- horses: ArrayList<Horse>
- injuredHorses: ArrayList<Horse>
- trainers: ArrayList<Trainer>
- jockeys: ArrayList<Jockey>
- sickJockeys: ArrayList<Jockey>
- cash: int
- rent: int
- salaries: int
+ week()
+ buy(Horse)
+ sell(Horse)
+ hire(Employee)
+ fire(Employee)
+ addObserver(PropertyChangeListener)
+ removeObserver(PropertyChangeListene
Factory Pattern: Employee creation is delegated to different factories based on what type of employee is beign created
<<abstract>> empFactory
+ create(String): Employee
<<abstract>> Employee
- name: string
- speed: int
- endurance: int
- healthy: boolean
+ getSpeed(): int
+ getEndurance(): int
jockeyFactory
+ create(String): Employee
trainerFactory
+ create(String): Employee
creates
Trainer
- trainingSkill: int
+ train(Horse, int)
trains
Horse
- name: string
- recoverTime: int
- healthy: boolean
- age: int
- speed: int
- endurance: int
- trainable: int
- trainer: Trainer
- stable: stable
+ train(Trainer)
+ getSpeed(): int
+ getEndurance(): int
+ getName(): String
+ updateStat(String)
Jockey
- style: RidingStyle
+ ride(Horse): RiddenHorse
+ getStyle(): String
rides
Strategy Pattern: Each jockey has a RidingStyle that defines how they can change a horse's attributes during a race
<<abstract>> RidingStyle
+ ride(Horse): RiddenHorse
+ style(): String
AggressiveRiding
+ ride(Horse): RiddenHorse
+ style(): String
BalancedRiding
+ ride(Horse): RiddenHorse
+ style(): String
ConservativeRiding
+ ride(Horse): RiddenHorse
+ style(): String
RiddenHorse
- jockey: Jockey
- horse: Horse
- speedBonus: Int
- endBonus: Int
+ addSpeed(int)
+ addEnd(int)
+ runDist(int)
+ getDistRun(): int
+ compareTo(): Comparator <RiddenHors
+ getJockName(): String
is ridden by
Decorator Pattern: Jockey acts as an "add-on" to horse's stats during races
Race
- name: string
- racers: ArrayList<RiddenHorse>
- length: int
+ run()
+ addRacer(RiddenHorse)
+ awardPrizes()
13, 26
horseFactory
+ create(): Horse
creates
GUI Handler. Uses Java's Swing library for implementation
Display
- frame: JFrame
- screen: String
- panel: JPanel
- header: JPanel
- footer: JPanel
- stable: Stable
+ draw()
+update(String)
+refresh()
+ menu()
+ start()
+ mainScreen()
+ horses()
+ race()
+ employees()
Observer Pattern: event compiler records events such as injuries and sickness. These evnts are stored and dispalyed for an in game week
PropertyChangeListener
implements
eventCompiler
- ArrayList<String> events
+ propertyChange(PropertyChangeEv
+ clear()
+ getEvent(): String
observes
Stable

Old Diagram


Sim
- stables: ArrayList<Stable>
- races: ArrayList<Race>
- horseCreator: horseFactory
- trainerCreator: empFactory
- jockeyCreator: empFactory
- injuryEvents: eventFirer
- sickEvents: eventFirer
- otherEvents: eventFirer
- year: int
- week: int
+ nextWeek()
+ summary()
Observer Pattern: eventfirer will occasionally initiate events like a horse being injured, employee being sick, or stables suddenly gaining or losing money. Stables, horses, and employees can subscribe to them to be updated
eventFirer
- observable: PropertyChangeSupport
+ fireEvent(String)
+ addObserver(Observer)
+ removeObserver(Observer)
subscribes
Horse
Employee
Stable
PropertyChangeListener
implements
Horse
Employee
Stable
Stable
- name: string
- horses: ArrayList<Horse>
- trainers: ArrayList<Trainer>
- jockeys: ArrayList<Jockey>
- foodType: int
- equipType: int
+ fire(Employee)
+ hire(Employee)
+ sell(Horse)
+ buy(Horse)
+ propertyChange(PropertyChangeEvent)
Factory Pattern: Employee creation is delegated to different factories based on what type of employee is beign created
<<abstract>> empFactory
+ create(String): Employee
<<abstract>> Employee
- name: string
- speed: int
- endurance: int
- acceleration: int
+ propertyChange(PropertyChangeEvent)
jockeyFactory
+ create(String): Employee
trainerFactory
+ create(String): Employee
creates
Trainer
- trainingSkill: int
+ train()
trains
Horse
- name: string
- imgFile: string
- tired: boolean
- recoverTime: int
- age: int
- speed: int
- endurance: int
- acceleration: int
- trainable: int
+ train(Trainer)
+ propertyChange(PropertyChangeEvent)
Race
- name: string
- racers: ArrayList<RiddenHorse>
- length: int
+ run()
13, 26
horseFactory
+ create(String): Horse
creates
Jockey
- style: RidingStyle
+ ride(Horse): RiddenHorse
Strategy Pattern: Each jockey has a RidingStyle that defines how they can change a horse's attributes during a race
<<abstract>> RidingStyle
+ ride(Horse): RiddenHorse
AggressiveRiding
+ ride(Horse): RiddenHorse
BalancedRiding
+ ride(Horse): RiddenHorse
ConservativeRiding
+ ride(Horse): RiddenHorse
RiddenHorse
- jockey: Jockey
- horse: Horse
+ race(Race)
rides
is ridden by
Decorator Pattern: Jockey acts as an "add-on" to horse's stats during races

Note that the new diagram does not include many of the getters and setters implemented in the final implementation, as those are not core to the design and don't serve much purpose in being in the final diagram.

The most important change from the original class diagram to the final was the structure of the Observer pattern. Originally, we wanted one subject that would "fire" events that would be received by horses and employees to cause injuries and the like. However, in our final implementation we found it more convenient to use one observer that recorded all events, which were outputted by the user's Stable object. This was best since we wanted a section of our main screen's UI to be a "news" section compiling events, thus it was easiest to have them stored as strings in one place so that we could pull them later and display them on the UI. Beyond that, the other changes were mostly to aid in smoother implementation, especially with the GUI - for example running races in the Simulation became separate from nextWeek since we had an individual dialog box to display the race results. We also removed several attributes to clean up some objects in our final design - for example, horse acceleration was removed since it was redundant with how we used speed. We also had to add some attributes in several classes to handle things like injury and sickness for horses and jockeys (specifically adding some arrays to Stable and health attributes to Jockeys and Horses). Overall, however, our general design remained fairly consistent from the beginning to the end.

Third-Party code vs. Original code Statement

The code for our project is original however there were instances where we used our past project and past lectures as resources for this new project.

Spencer did find help when figuring out how to implement the dialog boxes in our GUI and the URL is provided below.

https://stackoverflow.com/questions/6555040/multiple-input-in-joptionpane-showinputdialog

Statement on the OOAD process for your overall Semester Project

- Creating the GUI was difficult as this was the first time either of us had to create one. This was hard both in terms of actually implementing the code and in terms of figuring out how to design it. We eventually went with one object that had multiple different functions depending on which screen needed to be displayed, but it is unclear if this is the best way to do that.
- An issue experienced was that I, Jairo Guzman, at times was not available to keep making updates to our code as there were situations outside of our project where my attention was needed.
- One positive is that thanks to the experience from this semester's projects some of the design elements naturally suggested themselves - for example, the use of a strategy pattern for a jockey's riding style or the use of a decorator to "add on" the jockey to a

horse to create a RiddenHorse for racing felt natural after we learned about them in class.

- Another issue with the design process was, like mentioned in the class diagram section, the structure of our observer pattern. While the original design was probably more in line with how we'd been taught to implement it, that didn't work too much with the aspects of the UI we wanted to include, so we ended up going with our final version instead.

Project Demo Link:

https://drive.google.com/file/d/1GLxI-N9VNRItWZJ0o1BUccRdKB0XelsU/view?usp=sharing