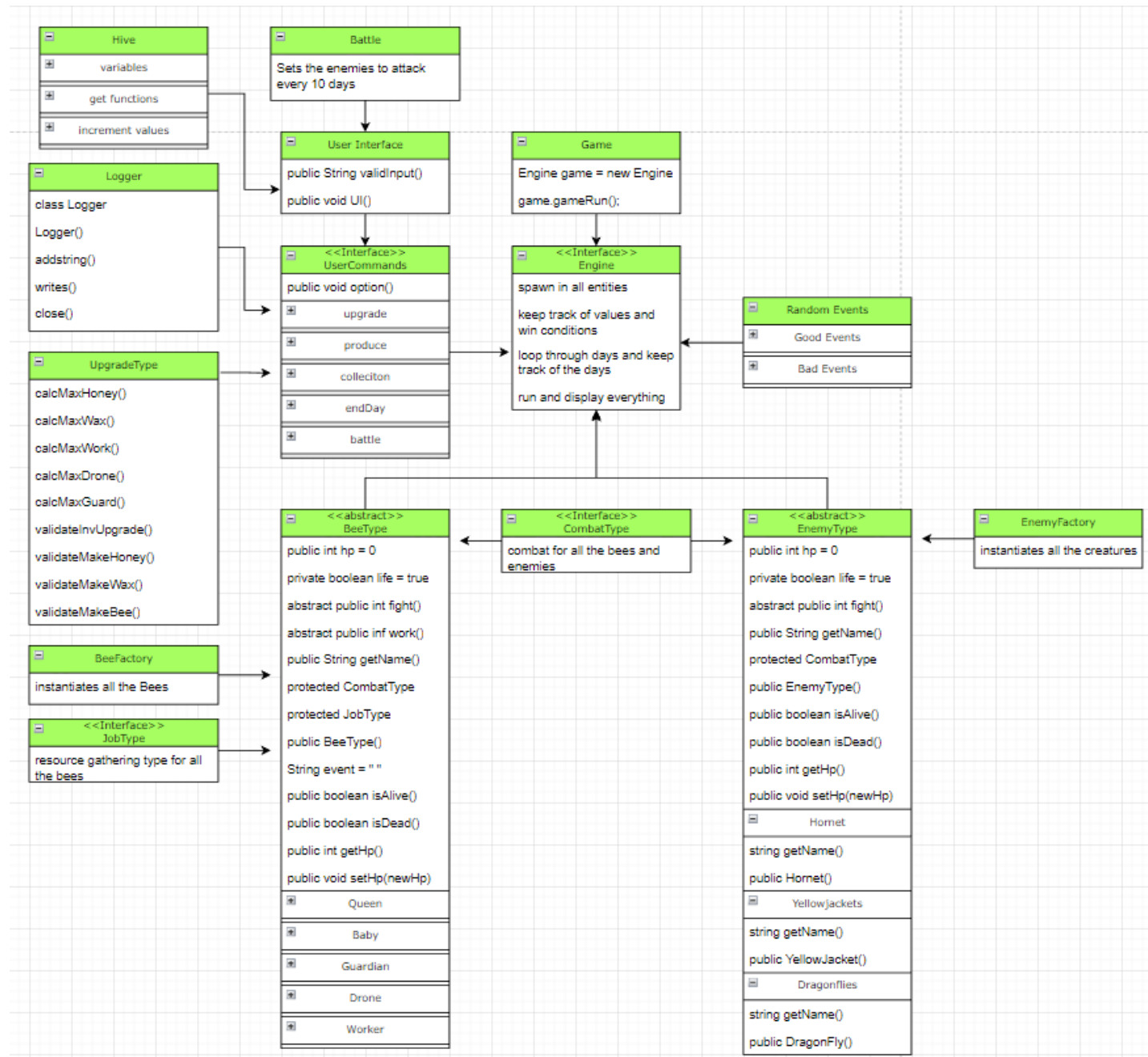Gabriel Barratt
Darian Payan

<u>Bee Town</u>

## Final State of our System:

Overall, our project didn't change much from the project 6 update. We are able to play our game with little to no issues and our code hits all of the requirements for the final. Comparing what we have now to what we originally planned, we have run into many more problems and edge cases as well as different interpretations on what we were trying to achieve. It was fun to bounce ideas off of each other and come to one conclusive final product compared to reading and interpreting a given set of instructions. From project 5 to project 6, we cleaned up a lot of our code as well as created more files for extra organization and clarity. This included adding a combat, job, and upgrade type for our game. We also took out some files that we thought we would use but ended up scrapping. From project 6 to project 7, we fleshed out more of the game and ended up running into a lot of scenarios we didn't initially plan for. This caused us to slow down and tackle things step by step until everything eventually worked as we had wanted. Essentially, Bee Town is a resource management game where the player has to survive a certain amount of days while fighting off other insects and continuously upgrade themselves. Some of the features that are included in the game are a random amount of resources collected at the start of each day, unique jobs and multipliers for each kind of bee the user can create, random events, battles, inventory upgrades and a tracker to see everything you have. One thing that was taken out due to time constraints was the idea of adding baby bee's and having them grow up over time. Other than that, the only other things that were altered was how we go about upgrades, values of certain materials, and naming.
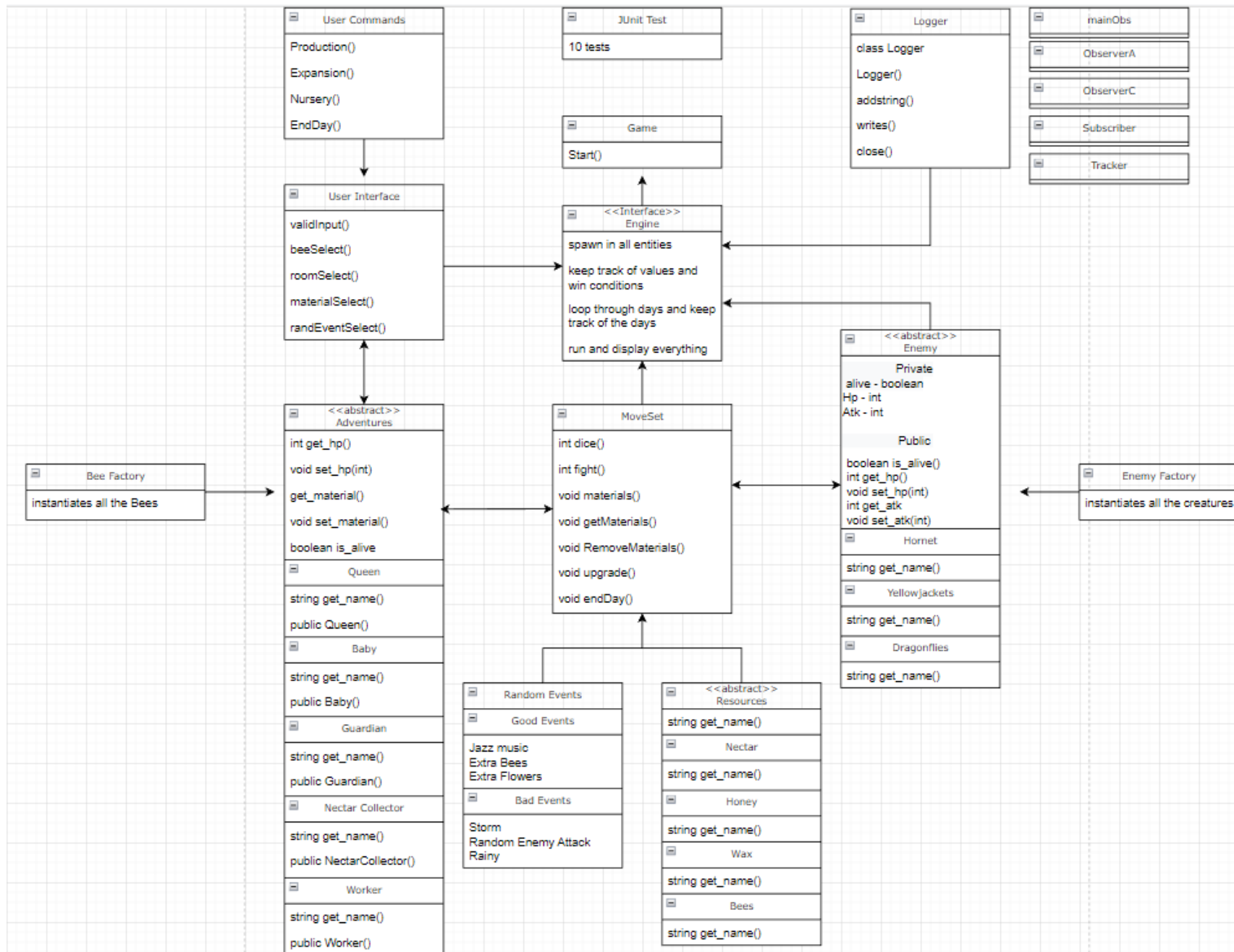
## Final Class Diagram: Patterns

- Command Pattern - Located in "UserCommands"
- Factory Pattern - All files with they key word "Factory"
- Decorator Pattern - All files with the key word "Type"
- Strategy Pattern - Located in "BeeType" and "EnemyType"
- Singleton Pattern - Logger

# Final Class Diagram

**Hive**
- variables
- get functions
- increment values

**Battle**
Sets the enemies to attack every 10 days

**User Interface**
- public String validInput()
- public void UI()

**Game**
- Engine game = new Engine
- game.gameRun();

**Logger**
- class Logger
- Logger()
- addstring()
- writes()
- close()

**<<Interface>> UserCommands**
- public void option()
- upgrade
- produce
- colleciton
- endDay
- battle

**<<Interface>> Engine**
- spawn in all entities
- keep track of values and win conditions
- loop through days and keep track of the days
- run and display everything

**Random Events**
- Good Events
- Bad Events

**UpgradeType**
- calcMaxHoney()
- calcMaxWax()
- calcMaxWork()
- calcMaxDrone()
- calcMaxGuard()
- validateInvUpgrade()
- validateMakeHoney()
- validateMakeWax()
- validateMakeBee()

**<> BeeType**
- public int hp = 0
- private boolean life = true
- abstract public int fight()
- abstract public inf work()
- public String getName()
- protected CombatType
- protected JobType
- public BeeType()
- String event = " "
- public boolean isAlive()
- public boolean isDead()
- public int getHp()
- public void setHp(newHp)
- Queen
- Baby
- Guardian
- Drone
- Worker

**<<Interface>> CombatType**
- combat for all the bees and enemies

**<> EnemyType**
- public int hp = 0
- private boolean life = true
- abstract public int fight()
- public String getName()
- protected CombatType
- public EnemyType()
- public boolean isAlive()
- public boolean isDead()
- public int getHp()
- public void setHp(newHp)
- **Hornet**
  - string getName()
  - public Hornet()
- **YellowJackets**
  - string getName()
  - public YellowJacket()
- **Dragonflies**
  - string getName()
  - public DragonFly()

**EnemyFactory**
- instantiates all the creatures

**BeeFactory**
- instantiates all the Bees

**<<Interface>> JobType**
- resource gathering type for all the bees

# Class Diagram Project 5

**User Commands**
- Production()
- Expansion()
- Nursery()
- EndDay()

**JUnit Test**
- 10 tests

**Game**
- Start()

**Logger**
- class Logger
- Logger()
- addstring()
- writes()
- close()

**mainObs**

**ObserverA**

**ObserverC**

**Subscriber**

**Tracker**

**User Interface**
- validInput()
- beeSelect()
- roomSelect()
- materialSelect()
- randEventSelect()

**<<Interface>> Engine**
- spawn in all entities
- keep track of values and win conditions
- loop through days and keep track of the days
- run and display everything

**<> Enemy**

Private
- alive - boolean
- Hp - int
- Atk - int

Public
- boolean is_alive()
- int get_hp()
- void set_hp(int)
- int get_atk
- void set_atk(int)

**Bee Factory**
- instantiates all the Bees

**<> Adventures**
- int get_hp()
- void set_hp(int)
- get_material()
- void set_material()
- boolean is_alive

**MoveSet**
- int dice()
- int fight()
- void materials()
- void getMaterials()
- void RemoveMaterials()
- void upgrade()
- void endDay()

**Enemy Factory**
- instantiates all the creatures

**Hornet**
- string get_name()

**Yellowjackets**
- string get_name()

**Dragonflies**
- string get_name()

**Queen**
- string get_name()
- public Queen()

**Baby**
- string get_name()
- public Baby()

**Guardian**
- string get_name()
- public Guardian()

**Nectar Collector**
- string get_name()
- public NectarCollector()

**Worker**
- string get_name()
- public Worker()

**Random Events**

**Good Events**
- Jazz music
- Extra Bees
- Extra Flowers

**Bad Events**
- Storm
- Random Enemy Attack
- Rainy

**<> Resources**
- string get_name()

**Nectar**
- string get_name()

**Honey**
- string get_name()

**Wax**
- string get_name()

**Bees**
- string get_name()

## Key Changes

Overall, the clarity and the organization of our project increased tremendously from project 5 to now. We added more decorators in order to keep the project organized and it allowed us to implement how bees collect nectar and fight off enemies. Getting rid of the moveset and instead splitting up what it would have done made our project flow more smoothly and it just made more sense to place certain things in separate files. Our understanding of the command pattern greatly improved from project 5 to now and that allowed us to fully utilize it in our game. We also changed the naming conventions and instead of upgrading rooms we decided to upgrade the inventory of each resource. A major change for our project was the addition of our Hive class. This is where we keep track of just about everything and change the values of our resources. It helps keep everything organized and it is easy to call and use all throughout the code.

## Third-Party code vs. Original code Statement

- Inspiration for the framework was taken originally taken from ROTLA, but was heavily tampered with and changed in the final version
- All code is original

## OOAD Process for the Semester Project

1. Factory Pattern - We found this pattern to be extremely helpful in keeping things organized and easy to understand. It is something that both of us will continue to use and implement in our code after this class.
2. Decorator Pattern - This was also majorly helpful in both our design and utility of our game. It not only saved us a lot of time, but it was also easy to use and implement. We both also hope to use and implement this in the future.
3. Planning during the design process - Our group could have done this better and maybe have fleshed it out more. While working together, we found ourselves talking about the same idea but had different understandings of what we wanted to do or accomplish with it. It would have been nice to plan things out more before starting the code because it would have saved us a decent amount of time.