* 1. **Class StarSystem**
     1. GOLDILOCK\_DISTANCE = 150,000,000 \\ goldilocks’ distance
     2. PREGZ\_RINGS = 2 // number of rings before goldilocks’ zone
     3. INGZ\_RINGS = 3 // number of rings in goldilocks’ zone
     4. POSTGZ\_RINGS = 6 // number of after before goldilocks’ zone
     5. +MIN\_PLANETS = 5
     6. +MAX\_PLANETS = 7
     7. +turns = 100 // == years\_till\_death\_star /10 (counts down)
     8. +index\_of\_main\_planet = NONE
     9. +planets\_list // All active planets (5-7)
     10. +distance\_list // list with all non GZ rings
     11. +gz\_distance\_list // list with all GZ rings
     12. +event : Event

In constructor:

* + 1. +set\_up() // set\_up method to generate planets must generate at least 1 goldilocks’ zone planets + all other planets
    2. +show\_planets() // show the user all generated planets
    3. +set\_main\_planet() // choose MainPlanet from planets\_list, use copy-constructor to copy all information to a new object of type MainPlanet; initialise index\_of\_main\_planet
    4. check\_winning\_condition() // **if** MainPlanet.engineering == 30 : Simulation won (final message) **else** **if** (turns != 0 && MainPlanet.engineering != 30) || MainPlanet.actual\_population <= 1 : Simulation over (final message)
    5. next\_turn() // decreases turns by 1 and offers the user information and the option to continue to the next turn
    6. +event.generate\_event(progression) // in the object event a new event is chosing from the list// frequency of generating an event per turn is random (e.g. 1/3)
    7. MainPlanet.change\_base\_values(event) //changes the values of the planet depending on the event
  1. **Class Planet()** 
     1. +Planet(distance) // constructor
     2. +landmass // value between 10 and 100
     3. +distance // distance from star to planet (get from set\_up)
     4. +atmosphere // composition atmosphere between 1 and 100 (100: perfect)
     5. +radius // radius of the planet
     6. +calc\_area() // calculates the area of a planet and returns it
     7. +calc\_temperature () // calculate temperature from **distance**, **atmosphere**
     8. + planet\_quality() // calculate QOL from return value calc\_temperature and landmass
     9. +show\_information()
  2. **Class MainPlanet(Planet)**
     1. +total\_population // amount of species
     2. +FOOD\_PER\_ORGANISM // constant, minimal need for one organism to be alive = 1
     3. +TECH\_CAP //constant, used to cap a certain tech
     4. +population\_health // state of organism
     5. +tech\_list // list of tech containing:
        1. +tech\_medicine;
        2. +tech\_agriculture;
        3. +tech\_architecture;
        4. +tech\_engineering;
     6. +tech\_focus //index of which tech has focus
     7. +tech\_counter //counter of how far the tech research is
     8. +spend\_points() // user spends points on technologies(4)
     9. +calc\_progression() // via tech & actual\_population (Jelle)
     10. set\_research\_focus() // choose your tech focus and generate +1 every 5 turns + creative input + show how many turns left till +1, when a new tech is chosen, the tech\_counter becomes 0
     11. +calc\_usable\_landmass() // calculate usable landmass with landmass, tech\_agriculture and tech\_architecture
     12. +change\_base\_values(generate\_event() = NONE) // updates:

*If* event.type\_event == 0 : disaster:

+update\_variables(event.get\_multipliers()) // multipliers\*variables // (variables = atmosphere, landmass, population\_health)

*If* event.type\_event == 1 : breakthrough update\_technologies(event.get\_tech\_index()) // immediate progress for a certain tech, unless tech is already max

landmass -> affects get\_usable\_landmass

atmosphere -> affects get\_tempeature();

life\_quality() -> affected by get\_usable\_landmass() and get\_temperature() and population\_health

total\_population -> affected by life\_quality()

* 1. **Class Event**
     1. +type\_event = NONE
     2. +atmosphere\_multiplier
     3. +landmass\_multiplier
     4. +population\_health\_multiplier
     5. +progression
     6. +title //string with the title
     7. +message //string with the message
     8. +tech\_index //if breakthrough than this decides which tech gets +1 (is dependent on which breakthrough), this is the index
     9. +disaster\_list // dict of lists (list contains all texts) – optional: read through file
     10. +breakthrough\_list // dict of lists (list contains all texts) – optional: read through file
     11. +Event(progression) // constructor // checks progression (type of disaster depends on this); generates either a disaster or a breakthrough // type\_event = 0 (disaster) or 1 (breakthrough), Does a higher progression mean a higher chance of getting a disaster?
     12. set\_multipliers // only if disaster // if unchanged (== not relevant) then multiplier = 1, else change multipliers
     13. update\_multipliers(tech : int[4]) //depending on the current tech of the main planet, the multipliers become less severe or worse
     14. get\_type() // returns type for the main planet to check
     15. get\_multipliers() // returns multipliers for the main planet to change its variables
     16. get\_tech\_index() //gives the index of which tech gets +1
     17. get\_title() //returns title
     18. get\_message() //returns message
     19. +generate\_event(progression)
  2. **Class Star(StarSystem) // optional: only when adding visual**
     1. +star\_size