# PSE Molekulardynamik

Sheet 1
First steps towards a molecular dynamics simulation

# Task 2: Completion of the program frame

## Implementation of basic algorithms

- Implementation of the formulas for force, position and velocities according to the script
- Particles updated with setter functions
  - setter for force also updates old_f
- Equality checking via == operation of Particle
- Value for each dimension is calculated separately

$$x_i(t_{n+1}) = x_i(t_n) + \Delta t \cdot v_i(t_n) + (\Delta t)^2 \frac{F_i(t_n)}{2m_i}$$

$$v_i(t_{n+1}) = v_i(t_n) + \Delta t \frac{F_i(t_n) + F_i(t_{n+1})}{2m_i}$$

$$F_i = \sum_{j=1, j\neq i}^{\#\text{particles}} F_{ij}$$

$$F_{ij} = \frac{m_i m_j}{(||x_i - x_j||_2)^3}(x_j - x_i)$$

# Task 2: Completion of the program frame

**Implementation of VTKWriter**

- InitializeOutput() for four particles to be plotted
- For each of the four particles call plotParticle()
- To write the particles to a file call writeFile()
- For each iteration, this procedure has to be repeated

# Task 3: Simulation of Halley's Comet

**Overview celestial bodies**

Setup: endtime: 250, delta_t: 0.01

# Task 3: Simulation of Halley's Comet

**Simulation in action**

# Task 3: Simulation of Halley's Comet

**Simulation with default_time problem**

# Task 4: Refactoring and Documentation

**Implementation of the ParticleContainer**

- Software pattern we used: iterator pattern

  → enables usage of range-based for-loop

  → easy iteration over particles and pairs of particles

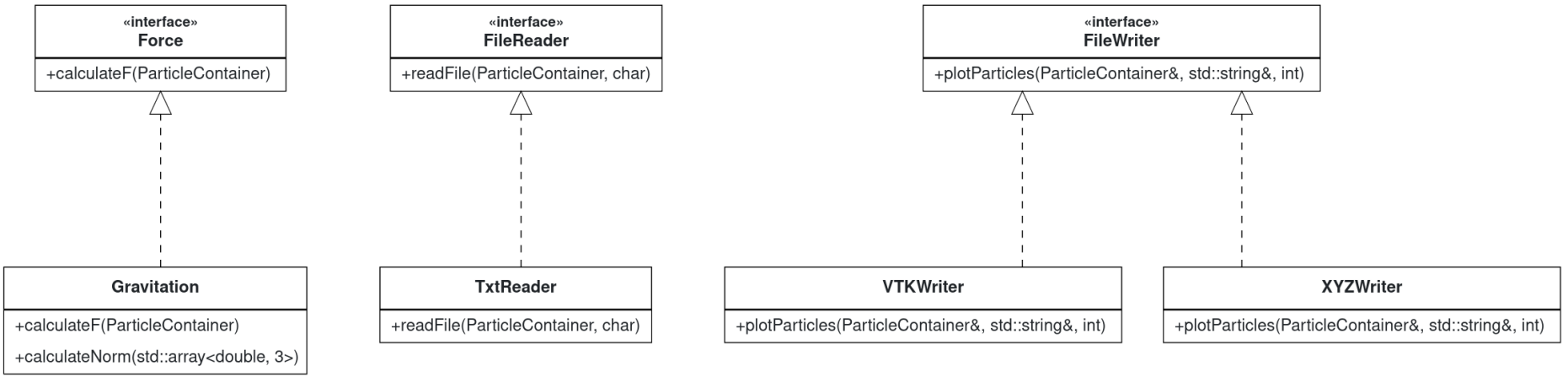
- Implementation as innerclass of ParticleContainer

  → begin(), end(), !=, ++, ==, *

- std::vector is used internally

# Task 4: Refactoring and Documentation

**Template pattern for I/O and calculation methods**




→ usage of std::unique_ptr with interfaces as static type and realization as dynamic type
→ usage of virtual destructor to prevent memory leaks

# Task 4: Refactoring and Documentation

**Datastructure for storing particles**

- ParticleContainer uses std::vector internally
- Mainly iteration over stored particles

  → vector uses a continous memory space

  → lower memory overhead when using vector

# Task 4: Refactoring and Documentation

**Doxygen and CMake**

Doxygen is useful for documenting the API, but not the implementation

Example:

```
/**
     * @brief Calculates Euclidean norm for three dimensional vector.
     *
     * @param x array of size three representing a three dimensional vector.
     *
     * @return Euclidian norm of passed vector.
    */
```

Integration in CMake:

- option DISABLE_DOXYGEN (OFF per default)
- Can be alter using -D DISABLE_DOXYGEN=ON