src/src/group.js

```
/**
 * @file Describes ADSODA group
 * @author Jeff Bigot <jeff@raktres.net> after Greg Ferrar
 * @class Group
 * @todo use it
 */

import { NDObject } from './ndobject.js'
import { v4 as uuidv4 } from 'uuid'

class Group extends NDObject {
  constructor (objects) {
    super('Group')
    this.space = ''
    this.objectList = new Set()
    this.uuid = uuidv4()
    if (objects) this.objectList = objects
  }

  /**
   * @returns JSON face description
   */
  exportToJSON () {
    return `{ "group" : ${JSON.stringify(this.objectList)} }`
  }

  /**
   *
   * @param {*} json
   */
  static importFromJSON (json, space) {
    const grp = new Group()
    const solids = new Map()
    space.solids.forEach(sol => {
      if (sol.id) {
        solids.set(sol.id, sol.uuid)
      }
    })
    json.refs.forEach(sol => {
      const solUuid = solids.get(sol)
      grp.objectList.add(solUuid)
    })
    grp.space = space
    return grp
  }

  /**
   * @returns text face description
   */
  logDetail () {
    return `Group name : ${this.name} \n --- objects : ${
      this.objectList
    } \n `
  }

  /**
   *
   */
  emptyGroup () {
    this.objectList.length = 0
  }

  /**
   * translate the face following the given vector.<br>
   * Translation doesn't change normal vector, Just the constant term need to be changed.
   * new constant = old constant - dot(normal, vector)<br>
   * @param {*} vector the vector indicating the direction and distance to translate this halfspace.
   * @todo vérifrie que mutation nécessaire
   * @returns face this
   */
  translate (vector) {
    // TODO: add selected control
    this.objectList.forEach(idx => {
      const object = this.space.solids.get(idx)
      object.translate(vector)
    })
    return this
  }

  /**
   *  This method applies a matrix transformation to this Halfspace.
   *  @param {matrix} matrix the matrix of the transformation to apply.
   * @todo vérifrie que mutation nécessaire
   * @returns face this
   */
  transform (matrix, center) {
    this.objectList.forEach(idx => {
      const object = this.space.solids.get(idx)
      object.transform(matrix, center)
    })
    return this
  }

  /**
   * @todo write
   */
  middleOf () {
    const dim = this.space.dimension
    const minCorner = []
    const maxCorner = []
    this.objectList.forEach(idx => {
      const object = this.space.solids.get(idx)
      object.corners.forEach(corner => {
        for (let i = 0; i < dim; i++) {
          minCorner[i] = Math.min(corner[i], minCorner[i] || corner[i])
          maxCorner[i] = Math.max(corner[i], maxCorner[i] || corner[i])
        }
      })
    })
    const corners = []
    for (let i = 0; i < dim; i++) {
      corners[i] = (maxCorner[i] + minCorner[i]) / 2
    }
    return corners
  }

  /**
   *
   */
  addObject (obj) {
    this.objectList.push(obj)
  }

  /**
   *
   */
  removeObject (obj) {
    // TODO:
  }
}
export { Group }

```