```
/*---------------------------------------------------------------------------*\
  =========                 |
  \\      /  F ield         | OpenFOAM: The Open Source CFD Toolbox
   \\    /   O peration     |
    \\  /    A nd           | Copyright (C) 2011-2013 OpenFOAM Foundation
     \\/     M anipulation  |
-------------------------------------------------------------------------------
License
    This file is part of OpenFOAM.

    OpenFOAM is free software: you can redistribute it and/or modify it
    under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    OpenFOAM is distributed in the hope that it will be useful, but WITHOUT
    ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or
    FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public License
    for more details.

    You should have received a copy of the GNU General Public License
    along with OpenFOAM.  If not, see <http://www.gnu.org/licenses/>.

Class
    Foam::myIncompressibleTwoPhaseMixture

Description
    A two-phase incompressible transportModel

SourceFiles
    myIncompressibleTwoPhaseMixture.C

\*---------------------------------------------------------------------------*/

#ifndef myIncompressibleTwoPhaseMixture_H
#define myIncompressibleTwoPhaseMixture_H

#include "incompressible/transportModel/transportModel.H"
#include "incompressible/viscosityModels/viscosityModel/viscosityModel.H"
#include "twoPhaseMixture.H"
#include "IOdictionary.H"


// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

namespace Foam
{

/*---------------------------------------------------------------------------*\
                      Class myIncompressibleTwoPhaseMixture Declaration
\*---------------------------------------------------------------------------*/

class myIncompressibleTwoPhaseMixture
:
    public IOdictionary,
    public transportModel,
    public twoPhaseMixture
{
protected:

    // Protected data

        autoPtr<viscosityModel> nuModel1_;
        autoPtr<viscosityModel> nuModel2_;

        dimensionedScalar rho1_;
        dimensionedScalar rho2_;
```

```
// ADDITION
        dimensionedScalar cp1_;
        dimensionedScalar cp2_;
        dimensionedScalar Pr1_;
        dimensionedScalar Pr2_;
// END of ADDITION
```

定圧比熱（cp）とプラントル数（Pr）の宣言を追加

```
        const volVectorField& U_;
        const surfaceScalarField& phi_;

        volScalarField nu_;


    // Private Member Functions

        //- Calculate and return the laminar viscosity
        void calcNu();


public:

    // Constructors

        //- Construct from components
        myIncompressibleTwoPhaseMixture
        (
            const volVectorField& U,
            const surfaceScalarField& phi
        );


    //- Destructor
    virtual ~myIncompressibleTwoPhaseMixture()
    {}


    // Member Functions

        //- Return const-access to phase1 viscosityModel
        const viscosityModel& nuModel1() const
        {
            return nuModel1_();
        }

        //- Return const-access to phase2 viscosityModel
        const viscosityModel& nuModel2() const
        {
            return nuModel2_();
        }

        //- Return const-access to phase1 density
        const dimensionedScalar& rho1() const
        {
            return rho1_;
        }

        //- Return const-access to phase2 density
        const dimensionedScalar& rho2() const
        {
            return rho2_;
        };

// ADDITION
        //- Return const-access to phase1 cp
        const dimensionedScalar& cp1() const
        {
            return cp1_;
        }
```

定圧比熱（cp）とプラントル数（Pr）をクラス外から取得するための関数を宣言

```
        //- Return const-access to phase2 cp
        const dimensionedScalar& cp2() const
        {
            return cp2_;
        }
        //- Return const-access to phase1 Pr
        const dimensionedScalar& Pr1() const
        {
            return Pr1_;
        }
        //- Return const-access to phase2 Pr
        const dimensionedScalar& Pr2() const
        {
            return Pr2_;
        }
// END of ADDITION

        //- Return the dynamic laminar viscosity
        tmp<volScalarField> mu() const;

        //- Return the face-interpolated dynamic laminar viscosity
        tmp<surfaceScalarField> muf() const;

// ADDITION
        //- Return the face-interpolated thermal conductivity
        tmp<surfaceScalarField> kappaf() const;
// END of ADDITION
```

セル界面での熱伝導率を求める関数 kappaf()を宣言

```
        //- Return the kinematic laminar viscosity
        virtual tmp<volScalarField> nu() const
        {
            return nu_;
        }

        //- Return the laminar viscosity for patch
        virtual tmp<scalarField> nu(const label patchi) const
        {
            return nu_.boundaryField()[patchi];
        }

        //- Return the face-interpolated kinematic laminar viscosity
        tmp<surfaceScalarField> nuf() const;

        //- Correct the laminar viscosity
        virtual void correct()
        {
            calcNu();
        }

        //- Read base transportProperties dictionary
        virtual bool read();
};


// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

} // End namespace Foam

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

#endif

// ************************************************************************* //
```