How to implement your own application

- The applications are located in the `$WM_PROJECT_DIR/applications` directory.
- Copy an application that is similar to what you would like to do and modify it for your purposes. In this case we will make our own copy of the `icoFoam` solver and put it in our `$WM_PROJECT_USER_DIR` with the same file structure as in the OpenFOAM installation:

```
cd $WM_PROJECT_DIR
cp -riuv --parents --backup applications/solvers/incompressible/icoFoam \
                                   $WM_PROJECT_USER_DIR
cd $WM_PROJECT_USER_DIR/applications/solvers/incompressible
mv icoFoam myIcoFoam
cd myIcoFoam
wclean
mv icoFoam.C myIcoFoam.C
```

- Modify `Make/files` to:

```
myIcoFoam.C
EXE = $(FOAM_USER_APPBIN)/myIcoFoam
```

- Compile with `wmake` in the `myIcoFoam` directory. `rehash` if necessary.

A look inside icoFoam

- The `icoFoam` directory consists of the following:

```
createFields.H   FoamX/   Make/   icoFoam.C
```

- The `FoamX` directory is of no interest unless you use FoamX. We will not care about that here.
- The `Make` directory contains instructions for the `wmake` compilation command.
- `icoFoam.C` is the main file, and `createFields.H` is an inclusion file, which is included in `icoFoam.C`.
- In the header of `icoFoam.C` we include `fvCFD.H`, which contains all class definitions that are needed for `icoFoam`. `fvCFD.H` is included from (see `Make/options`) `$WM_PROJECT_DIR/src/finiteVolume/lnInclude`, but that is actually only a link to `$WM_PROJECT_DIR/src/finiteVolume/cfdTools/general/include/fvCFD.H`. `fvCFD.H` in turn only includes other files that are needed (see next slide).
- Hint: Use `find PATH -iname "*LETTERSINFILENAME*"` to find where in `PATH` a file with a file name containing `LETTERSINFILENAME` in its file name is located.
  In this case: `find $WM_PROJECT_DIR -iname "*fvCFD.H*"`

A look inside icoFoam, fvCFD.H

```
#ifndef fvCFD_H
#define fvCFD_H

#include "parRun.H"

#include "Time.H"
#include "fvMesh.H"
#include "fvc.H"
#include "fvMatrices.H"
#include "fvm.H"
#include "linear.H"
#include "calculatedFvPatchFields.H"
#include "fixedValueFvPatchFields.H"
#include "adjustPhi.H"
#include "findRefCell.H"
#include "mathematicalConstants.H"
```

```
#include "OSspecific.H"
#include "argList.H"

#ifndef namespaceFoam
#define namespaceFoam
    using namespace Foam;
#endif

#endif
```

The inclusion files are all class definitions that are used in `icoFoam`. Dig further into the source file to find out what these classes actually do.
At the end we say that we will use all definitions made in `namespace Foam`.

A look inside icoFoam

- `icoFoam` starts with

  ```
  int main(int argc, char *argv[])
  ```

  where `int argc, char *argv[]` are the number of parameters, and the actual parameters used when running `icoFoam`.

- The case is initialized by:

  ```
  #    include "setRootCase.H"

  #    include "createTime.H"
  #    include "createMesh.H"
  #    include "createFields.H"
  #    include "initContinuityErrs.H"
  ```

  where all inclusion files except `createFields.H` are included from `src/OpenFOAM/lnInclude` and `src/finiteVolume/lnInclude`. Have a look at them yourself. (find them using the `find` command)

- `createFields.H` is located in the `icoFoam` directory. It initializes all the variables used in `icoFoam`. Have a look inside it and see how the variables are created from files.

A look inside icoFoam

- The time loop starts by:

```
for (runTime++; !runTime.end(); runTime++)
```

  and the rest is done at each time step.

- The `fvSolution` subdictionary `PISO` is read, and the Courant number is calculated and written to the screen by (use the `find` command)

```
#       include "readPISOControls.H"
#       include "CourantNo.H"
```

- The momentum equations are defined and a velocity predictor is solved by

```
fvVectorMatrix UEqn
(
    fvm::ddt(U)
  + fvm::div(phi, U)
  - fvm::laplacian(nu, U)
);

solve(UEqn == -fvc::grad(p));
```

A look inside icoFoam, the PISO loop

- A PISO corrector loop is initialized by

```
for (int corr=0; corr<nCorr; corr++)
```

- The member functions of the PISO algorithm are:
  (Descriptions taken from the classes of each object used when calling the functions)
  `A()`: Return the central coefficient of an `fvVectorMatrix`.
  `H()`: Return the H operation source of an `fvVectorMatrix`.
  `Sf()`: Return cell face area vectors of an `fvMesh`.
  `flux()`: Return the face-flux field from an `fvScalarMatrix`
  `correctBoundaryConditions()`: Correct boundary field of a `volVectorField`.

- Find the descriptions by identifying the object type (class) and then search the OpenFOAM Doxygen at: `http://foam.sourceforge.net/doc/Doxygen/html/` (linked to from `www.openfoam.org`).

- See *Rhie and Chow in OpenFOAM*, by Fabian Peng Kärrholm at the course homepage for a detailed description of the PISO algorithm and Rhie and Chow in OpenFOAM.

A look inside icoFoam, write statements

- At the end of icoFoam there are some write statements:

```
runTime.write();

Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
    << "  ClockTime = " << runTime.elapsedClockTime() << " s"
    << nl << endl;
```

- `write()` makes sure that all variables that were defined as an `IOobject` with `IOobject::AUTO_WRITE` are written to the time directory accoring to the settings in the `controlDict` dictionary.
- `elapsedCpuTime()` is the elapsed CPU time.
- `elapsedClockTime()` is the elapsed wall clock time.