Popolo Reference Manual

CC BY SA

Copyright (c) 2010 Eisuke Togashi
Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

Contents

Chapter 1 Introduction

1.1 What is Popolo

Popolo is a collection of classes for calculating various heat transfer phenomena. The routines have been written from scratch in C#, and present a modern Applications Programming Interface (API) for .NET Framework programmers, allowing wrappers to be written for very high level languages. It contains classes to calculate solid conduction, convective heat transfer near wall surfaces, air ventilation, radiative heat balance of wall surfaces, transmitted solar radiation through a window, and so on. Users should build up these classes to simulate a whole complex building system. A sample source code to build test cases of BESTEST are provided. Since all the source code is distributed under the GNU General Public License, they can be freely downloaded from the Web site.

This manual describes how to use Popolo in your program. Some example codes are also provided.

1.2 Where to Download Popolo

The latest release of Popolo can be downloaded from website (http://www.hvacsimulator.net). If you extract zipped file, you can find two dll files, Popolo.dll and GSLNET.dll. Popolo.dll is a main file and GSLNET.dll numerical library which is used in Popolo. GSLNET.dll is a wrapper library for GSL (GNU Scientific Library).

1.3 Making the first simple program

In order to use a “Popolo.dll” in your application, you must first add a reference to it. The procedure to make reference to dll files with Visual Studio 2008 is described below.

Figure 1.1 shows the start up window of Visual Studio. Selecting “File” - “New” - “Project”, you can open “New project” window as shown in Figure 1.2. Select “Visual C#” and “Console Application”, then click “OK” button.[†1)]

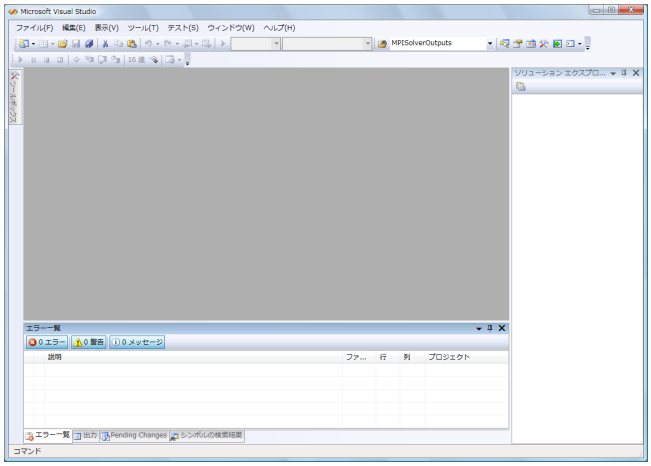


Fig.1.1 Start up window of Visual Studio

†1) You may also use some other languages which support .NET Framework such as C++.NET or Basic.NET.

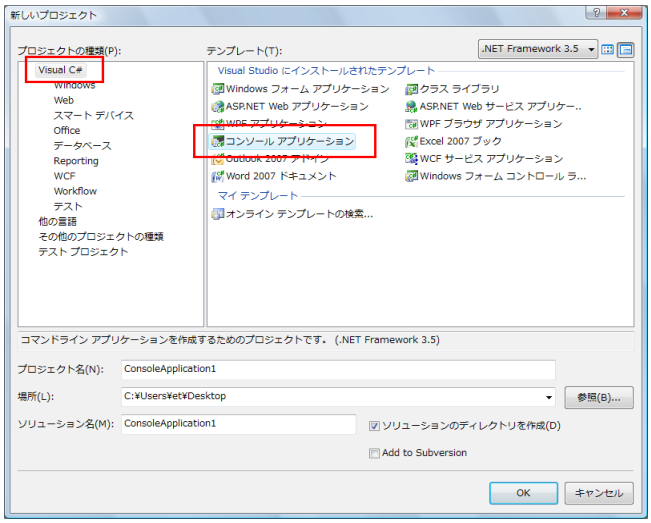


Fig.1.2 New Project Window

As shown in figure 1.3, a simple program is automatically generated by Visual Studio. In Solution Explorer, right-click on the project node and click Add Reference. In the Add Reference dialog box (Figure 1.4), select the "Browse" tab to browse for "Popolo.dll" in the file system (Figure 1.5). You can find that the reference to the "Popolo.dll" is added in the Solution Explorer (Figure 1.6).

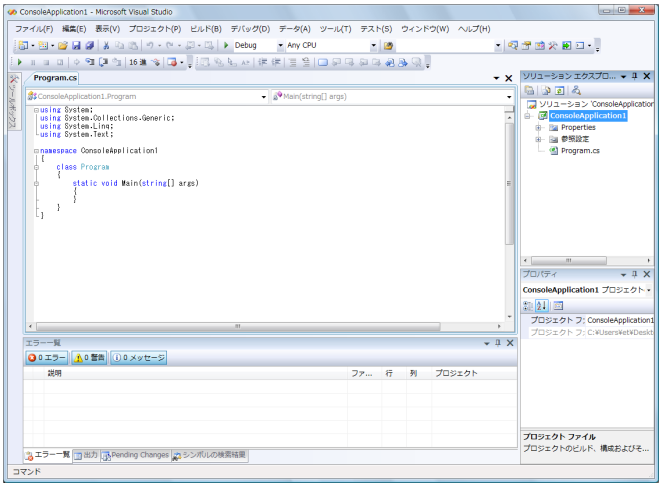


Fig.1.3 Console application project automatically generated by Visual Studio

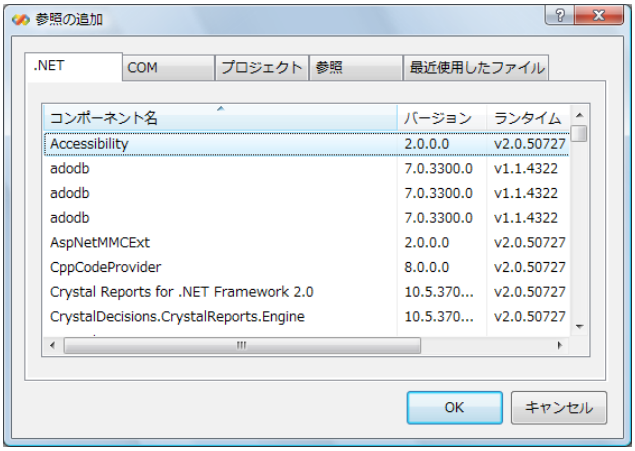


Fig.1.4 Add Reference dialog box 1

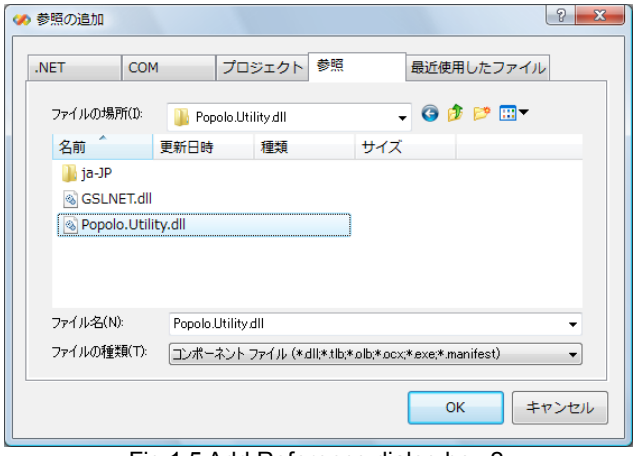


Fig.1.5 Add Reference dialog box 2

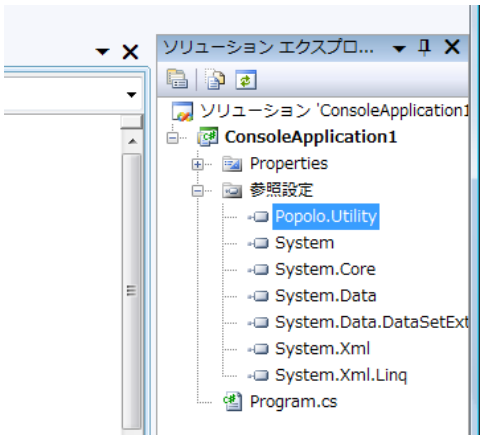


Fig.1.6 Reference to “Popolo.dll”

Popolo.dll includes various types of classes to execute building environmental simulation. They are divided into some name spaces. C# programs are organized using namespaces. “”using” directives are provided to facilitate the use of namespaces.

The following example (Figure 1.7) shows how to define a using directive. In line 3, “*Popolo*.*ThermophysicalProperty*” namespace is referenced. It is the namespace that provides classes to calculate thermo-physical property of water or moist air. In line 11, *MoistAir* class which belongs to “*Popolo*.*ThermophysicalProperty*” is called.

```
1  using System;
2
3  using Popolo.ThermophysicalProperty;
4
5  namespace ConsoleApplication
6  {
7     class Program
8     {
9        static void Main(string[] args)
10       {
11          double cpAir = MoistAir.GetSpecificHeat(0.018);
12          Console.WriteLine("Specific heat of moist air at humidity ratio of 0.018 kg/kg(DA) is" + cpAir.ToString(“F3”) + "kJ/K");
13          Console.Read();
14       }
15    }
16 }
```

Fig.1.7 Sample program using “Popolo.ThermophysicalProperty” namespace

Either by hitting F5 key or choosing "Start Debugging" from the menu, the executable will start. Figure 1.8 is the result of the program. You can find that the specific heat of moist air is successfully calculated

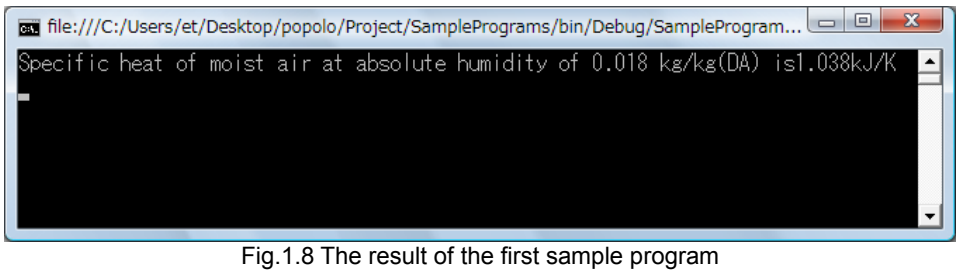


Fig.1.8 The result of the first sample program

Chapter 2 Calculating Thermodynamic Properties

2.1 Name space for calculating thermodynamic properties

The classes which calculate a thermodynamic properties belong to "*Popolo*.*ThermophysicalProperty*" namespace. Table 2.1 shows principal members defined in "*Popolo*.*ThermophysicalProperty*" namespace.

Table 2.1 Principal members defined in "Popolo.ThermophysicalProperty" namespace

Name	Type	General function
MoistAir	class	A class which express the moist air. The functions which calculates thermodynamic properties of the moist air is also defined in this class.
ImmutableMoistAir	interface	Read only interface for MoistAir class.
Water	static class	The functions which calculates thermodynamic properties of the water is defined in this class.

2.2 Calculating thermodynamic properties of moist air

The *MoistAir* class can be used to calculate thermodynamic properties of the moist air. Most of functions are transported from HVACSIM[1)]. and some of the functions are based on the Udagawa's program[2)].

2.2.1 The members of the MoistAir class

There are 7 primary variables which characterize a moist air, "*Drybulb* temperature", "*Wetbulb* temperature", "Enthalpy", "Relative humidity", "Humidity ratio", "Specific volume" and "Atmospheric pressure". These variables are defined as properties of the *MoistAir* class as shown in the table 2.2.

Table 2.2 Properties of the MoistAir class

Name	Mean	Has set accessor	Unit	Type
DryBulbTemperature	Drybulb temperature	yes	°C	double
WetBulbTemperature	Wetbulb temperature	yes	°C	double
Enthalpy	Enthalpy	yes	kJ/kg	double
RelativeHumidity	Relative humidity	yes	%	double
HumidityRatio	Humidity ratio	yes	kg/kg(DA)	double
SpecificVolume	Specific volume	yes	m^3/kg	double
AtmosphericPressure	Atmospheric pressure	yes	kPa	double

Figure 2.1 is the sample code which edits the values of the properties of *MoistAir* object. In the line 11, an instance of the *MoistAir* class is created. From line 13 to line 20, values are set to the properties. From line 22 to line 29, the values of the *MoistAir* object is written to the standard output stream.

```
1  using System;
2  using Popolo.ThermophysicalProperty;
3
4  namespace ConsoleApplication
5  {
6      class Program
7      {
8          static void Main(string[] args)
9          {
10             //Creating instance of MoistAir class
11             MoistAir mAir = new MoistAir();
12
13             //Set values to property
14             mAir.DryBulbTemperature = 25.6;
15             mAir.HumidityRatio = 0.018;
16             mAir.RelativeHumidity = 50.0;
17             mAir.WetBulbTemperature = 22;
18             mAir.SpecificVolume = 0.86;
19             mAir.Enthalpy = 58.0;
20             mAir.AtmosphericPressure = 101.325;
21
22             //Output values of properties
23             Console.WriteLine("Drybulb Temperature:" + mAir.DryBulbTemperature);
24             Console.WriteLine("Humidity Ratio:" + mAir.HumidityRatio);
25             Console.WriteLine("Relative Humidity:" + mAir.RelativeHumidity);
26             Console.WriteLine("Wetbulb Temperature:" + mAir.WetBulbTemperature);
27             Console.WriteLine("Specific Volume:" + mAir.SpecificVolume);
28             Console.WriteLine("Enthalpy:" + mAir.Enthalpy);
29             Console.WriteLine("Atmospheric Pressure:" + mAir.AtmosphericPressure);
30
31             Console.Read();
32         }
33     }
34 }
```

Fig.2.1 The sample code to edit the values of the MoistAir property

Figure 2.2 shows the result of the sample code 2.1.

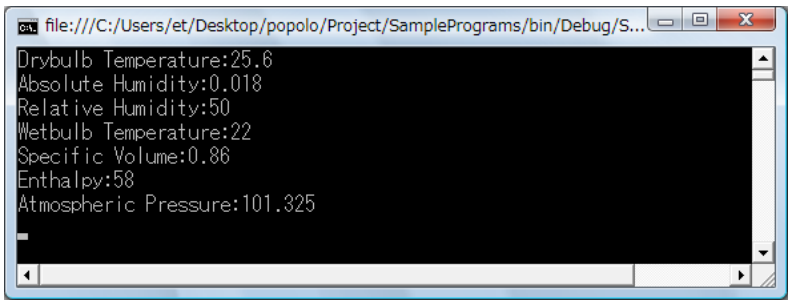


Fig.2.2 Result of the sample code 2.1

2.2.2 How to calculate thermodynamic properties of the moist air

An enumeration type "Property" which expresses the moist air properties is defined in *MoistAir* class. Table 2.3 shows a list of *MoistAir.Property* enumerator.

For each atmospheric pressure, two known properties allow determination of all other properties of the moist air. To calculate thermodynamic properties from two known properties, user can use static methods shown in table 2.4. The static methods are named as "*GetAirStateFromXXYY*", where *XX* and *YY* represent two known properties. DB is Dry-Bulb temperature, HR is Humidity Ratio, RH is Relative Humidity, EN is ENthalpy, WB is Wet-Bulb temperature, SV is Specific Volume.

Table 2.3 A list of MoistAir.Property enumerator

Name	Meaning
DryBulbTemperature	Drybulb Temperature[°C]
WetBulbTemperature	Wetbulb Temperature [°C]
HumidityRatio	Humidity Ratio [kg/kg(DA)]
RelativeHumidity	Relative Humidity [%]
Enthalpy	Enthalpy [kJ/kg]
WaterPartialPressure	Water Partial Pressure [kPa]
SpecificVolume	Specific Volume [m^3/kg]
SaturatedTemperature	Saturated Temperature [°C]

Table 2.4 Static methods to calculate thermodynamic properties of the moist air

Name	General function
GetAirStateFromAHEN	Calculate thermodynamic properties from humidity ratio [kg/kg] and enthalpy [kJ/kg]
GetAirStateFromAHRH	Calculate thermodynamic properties from humidity ratio [kg/kg] and relative humidity [%]
GetAirStateFromAHSV	Calculate thermodynamic properties from humidity ratio [kg/kg] and specific volume [m^3/kg]
GetAirStateFromDBAH	Calculate thermodynamic properties from dry bulb temperature [°C] and humidity ratio [kg/kg]
GetAirStateFromDBEN	Calculate thermodynamic properties from dry bulb temperature [°C] and enthalpy [kJ/kg]
GetAirStateFromDBRH	Calculate thermodynamic properties from dry bulb temperature [°C] and relative humidity [%]
GetAirStateFromDBSV	Calculate thermodynamic properties from dry bulb temperature [°C] and specific volume [m^3/kg]
GetAirStateFromDBWB	Calculate thermodynamic properties from dry bulb temperature [°C] and wet bulb temperature [°C]
GetAirStateFromRHEN	Calculate thermodynamic properties from relative humidity [%] and enthalpy [kJ/kg]
GetAirStateFromRHSV	Calculate thermodynamic properties from relative humidity [%] and specific volume [m^3/kg]
GetAirStateFromWBAH	Calculate thermodynamic properties from wet bulb temperature [°C] and humidity ratio [kg/kg]
GetAirStateFromWBEN	Calculate thermodynamic properties from wet bulb temperature [°C] and enthalpy [kJ/kg]
GetAirStateFromWBRH	Calculate thermodynamic properties from wet bulb temperature [°C] and relative humidity [%]
GetAirStateFromWBSV	Calculate thermodynamic properties from wet bulb temperature [°C] and specific volume [m^3/kg]

Each method is over loaded and have 4 combinations of parameters. Table 2.5 shows return value and 4 combinations of parameters.

The first and the second method calculates all the thermodynamic properties of the moist air from given two state. It returns *MoistAir* object whose Properties are correctly set upped. The second method request a value of atmospheric pressure as third parameter. By contrast, the first method suppose the value of the atmospheric pressure as 101.325 kPa.

The third and the fourth method calculates value of only one specific property of the moist air. They need shorter calculating time than the first and the second method.

Table 2.5 Return value and 4 combinations of parameters

No.	Return value	param 1	param 2	param 3	param 4
1	MoistAir object	value of specific property 1	value of specific property 2	N/A	N/A
2	MoistAir object	value of specific property 1	value of specific property 2	atmospheric pressure [kPa]	N/A
3	value of specific property (double)	value of specific property 1	value of specific property 2	MoistAir.Property	N/A
4	value of specific property (double)	value of specific property 1	value of specific property 2	MoistAir.Property	atmospheric pressure [kPa]

Figure 2.3 shows the sample code which calculates the values of the properties of *MoistAir*. In the line 14, using "*GetAirStateFromDBHR*" method, state of the moist air is calculated from the value of dry bulb temperature and humidity ratio. The value of the moist air are written to the standard output stream. In the line 27, "*MoistAir.Property*" enumerator is given as third parameter of the *GetAirStateFromDBEN* method. Therefore, only the value of humidity ratio is calculated.

Figure 2.4 shows the result.

```
1  using System;
2  using Popolo.ThermophysicalProperty;
3
4  namespace ConsoleApplication
5  {
6     class Program
7     {
8        static void Main(string[] args)
9        {
10          //Create an instance of the MoistAIr class
11          MoistAir mAir;
12
13          //Calculate state of the moist air from given two properties (DB 25 °C, AH 0.012 kg/kg)
14          mAir = MoistAir.GetAirStateFromDBAH(25, 0.012);
15
16          //Write value of the moist air to standard output stream.
17          Console.WriteLine("Dry bulb temperature:" + mAir.DryBulbTemperature.ToString("F1"));
18          Console.WriteLine("Humidity ratio:" + mAir.HumidityRatio.ToString("F3"));
19          Console.WriteLine("Relative humidity:" + mAir.RelativeHumidity.ToString("F1"));
20          Console.WriteLine("Wet bulb temperature:" + mAir.WetBulbTemperature.ToString("F1"));
21          Console.WriteLine("Specific volume:" + mAir.SpecificVolume.ToString("F3"));
22          Console.WriteLine("Enthalpy:" + mAir.Enthalpy.ToString("F1"));
23          Console.WriteLine("Atmospheric pressure:" + mAir.AtmosphericPressure.ToString("F1"));
24          Console.WriteLine();
25
26          //Calculate relative humidity from dry bulb temperature and enthalpy.
27          double rHumid = MoistAir.GetAirStateFromDBEN(25, 58, MoistAir.Property.RelativeHumidity);
28          Console.WriteLine("Relative Humidity:" + rHumid.ToString("F1"));
29
30          Console.Read();
31       }
32    }
33 }
```

Fig.2.3 The sample code calculating the values of the properties of moist air

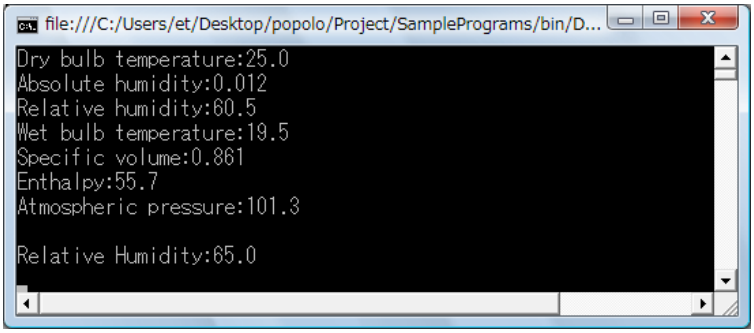


Fig.2.4 Result of the sample code

2.2.3 Calculation of saturated moist air

Table 2.6 shows the lists of static method to calculate saturated moist air state. They take two or three arguments. The first argument is value of a specific property of the moist air (ex. dry-bulb temperature, humidity ratio). The second argument is "*MoistAir.Property*", the enumerating object. The third object is value of atmospheric pressure.

Table 2.6 The lists of static method to calculate saturated moist air state

Name	General function
GetSaturatedDrybulbTemperature	Calculate saturated dry-bulb temperature from one of value of moist air property.
GetSaturatedHumidityRatio	Calculate saturated humidity ratio from one of value of moist air property.
GetSaturatedEnthalpy	Calculate saturated enthalpy from one of value of moist air property.
GetSaturatedVaporPressure	Calculate saturated vapor pressure from one of value of moist air property.

2.2.4 Immutable interface for MoistAir class

As described above, *MoistAir* object has double type properties which contains values of the moist air states (ex. dry-bulb temperature, humidity ratio). Therefore, *MoistAir* object could be an argument or return value of some other method. For example, when we calculate an air handling unit, a cooling coil need state of inlet moist air which is outlet state of a fan. In this case, a *MoistAir* object will be given from a Fan object to a *CoolingCoil* object. Since the cooling coil has no ability to alter the value of inlet moist air state, the access for properties of *MoistAir* objects should be denied from the *CoolingCoil* object.

To enable this access control, *ImmutableMoistAir* interface is provided. *ImmutableMoistAir* interface has exactly same properties as *MoistAir* class, but only get accessors are defined. Therefore, we can't set values to *ImmutableMoistAir* properties. They are read-only properties. MoistAir object can be treated as *ImmutableMoistAir* object, since it implements *ImmutableMoistAir* interface. Figure 2.5 is the UML diagrams.

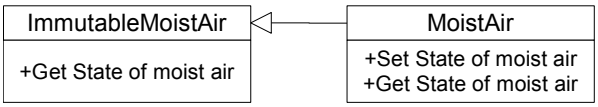


Fig.2.5 UML diagram of MoistAir class and ImmutableMoistAir interface

Figure 2.6 is a sample code which uses *ImmutableMoistAir* interface.

```
1  using System;
2
3  using Popolo.ThermophysicalProperty;
4
5  namespace ConsoleApplication
6  {
7     class Program
8     {
9        static void Main(string[] args)
10       {
11          //Create an instance of ImmutableMoistAir
12          ImmutableMoistAir mAir;
13
14          //Create MoistAir object and set it to ImmutableMoistAir object
15          mAir = MoistAir.GetAirStateFromDBAH(25, 0.012);
16
17          //Compile error occurs since ImmutableMoistAir interface doesn't have set accessors
18          //mAir.DryBulbTemperature = 27;
19          //mAir.Enthalpy = 56;
20
21       }
22    }
23 }
```

Fig. 2.6 Sample code which uses ImmutableMoistAir interface.

2.2.5 Other methods defined in MoistAIr class

1) BlendAir method

To calculate blended moist air state, *BlendAir* method is defined. Table 2.7 shows arguments of *BlendAir* method. Figure 2.7 is a sample code which uses *BlendAir* method. It creates two *ImmutableMoistAir* objects and blend them (mixing ratio is 3:7).

Table 2.7 Arguments of BlendAir method

No.	Argument 1	Argument 2	Argument 3	Argument 4
1	Blended moist air 1 (ImmutableMoistAir)	Blended moist air 2 (ImmutableMoistAir)	Blend rate of moist air 1 (double)	Blend rate of moist air 2 (double)
2	List of blended moist air (ImmutableMoistAir[ ])	Blend rate of moist air (double[ ])	-	-
3	Dry-bulb temperature list of blended moist air (double[ ])	Humidity raiot list of blended moist air (double[ ])	-	-

```
using System;

using Popolo.ThermophysicalProperty;

namespace ConsoleApplication
{
    class Program
    {
        static void Main(string[] args)
        {
            //Create an instance of MoistAir class
            ImmutableMoistAir mAir1, mAir2, blendAir;

            //Calculate moist air state (DB 25C, AH 0.012kg/kgDA)
            mAir1 = MoistAir.GetAirStateFromDBAH(25, 0.012);
            //Calculate moist air state (DB 30C, AH 0.014kg/kgDA)
            mAir2 = MoistAir.GetAirStateFromDBAH(30, 0.014);

            //Blend moist air. (Blend rate is 3:7)
            blendAir = MoistAir.BlendAir(mAir1, mAir2, 0.3, 0.7);

            Console.WriteLine("Dry bulb temperature:" + blendAir.DryBulbTemperature);
            Console.WriteLine("Humidty ratio:" + blendAir.HumidityRatio);
            Console.WriteLine("Relative humidity:" + blendAir.RelativeHumidity);
            Console.WriteLine("Wet bulb temperature:" + blendAir.WetBulbTemperature);
            Console.WriteLine("Specific volume:" + blendAir.SpecificVolume);
            Console.WriteLine("Enthalpy:" + blendAir.Enthalpy);
        }
    }
}
```

Fig. 2.7 Sample code which uses BlendAir method

2) GetDynamicViscosity method

Method to calculate dynamic viscosity [m^2/s] of moist air. An argument is dry bulb temperature.

3) GetSpecificHeat method

Method to calculate specific heat [kJ/(kg K)] of moist air. An argument is humidity ratio.

4) GetThermalConductivity method

Method to calculate thermal conductivity [W/(m K)] of moist air. An argument is dry bulb temperature.

5) GetWaterVaporPressure method

Method to calculate water vapor pressure [kPa] of moist air. An argument is humidity ratio (and atmospheric pressure).

Chapter 3 Calculating Circuit Network

3.1 Name space for calculating circuit network

The classes which calculate a circuit network belong to "*Popolo.CircuitNetwork*" namespace. Table 3.1 shows principal members defined in "*Popolo.CircuitNetwork*" namespace.

Table 3.1 Principal members defined in "Popolo.CircuitNetwork" namespace

Name	Type	General function
Channel	class	A class which expresses a channel.
Circuit	class	A class which expresses a circuit. It consists of Channel and Node object.
CircuitSolver	class	A class which solve a Circuit object.
ImmutableChannel	interface	Read only interface for Channel class.
ImmutableCircuit	interface	Read only interface for Circuit class.
ImmutableNode	interface	Read only interface for Node class.
Node	class	A class which expresses a node.

A potential at each node in circuit network can be expressed in equation 3.1[3]. To solve a circuit network, user should make a Circuit object with a Channel and a Node object, which expresses equation 3.1. Then, solve the Circuit object with a *CircuitSolver* object.

$$m \cdot \frac{dp}{dt} = \sum_{i=0}^{N} \frac{1}{R_i} \cdot (p_i - p)^{\frac{1}{\eta}} + G \tag{3.1}$$

m : Capacity of node p : Potential of node N : Number of nodes which are connected to target node
R_i : Resistance between nodes G : Energy flow from the outside to node η : resistive index ($1 \leq \eta \leq 2$)

3.2 General descriptions of classes in CircuitNetwork namespace

3.2.1 Node class

Node is a class which expresses a node. A principal parameters of node are potential and capacity. A variable p and m in equation 3.1 represent potential and capacity. If the capacity is very small, value of the potential changes rapidly with the energy flow. User can initialize these value in constructor. For example, to make Node object whose capacity and potential are 5 and 10, user should write code like below. The first argument is a name of the node, the second is value of the capacity, and the third is value of the potential.

```
Node sampleNode = new Node("SampleNode", 5, 10)
```

To get the list of the channels which are connected to the node, use *GetChannels* method. To get the total energy flow to node, use *GetTotalFlow* method. If the returned value of *GetTotalFlow* method is less than 0, energy flows to outside of the node.

If the node is the boundary node (the node whose potential should be kept constant value), set value of the *IsBoundaryNode* property to true. To make constant energy flow to the node, set the energy flow to the *ExternalFlow* property. If the *ExternalFlow* is positive value, it means that energy flows from node to outside. If negative, energy flows from outside to node. A variable G in equation 3.1 represent the *ExternalFlow.*

These properties are used when user make boundary condition. Fig.3.1 shows how to set the boundary conditions.

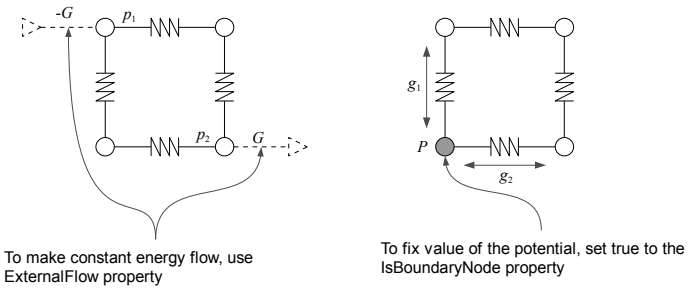


Figure 3.1 How to set the boundary conditions

As same as the *MoistAir* class, an immutable interface (*ImmutableNode*) for the Node class is defined. The Node class implements the *ImmutableNode* interface.

3.2.2 Channel class

Channel is a class which expresses a channel. It connects two node objects and represent their connection weight with resistance and resistive index. A variable R and η in equation 3.1 represent resistance and resistive index. User can initialize these value in constructor. For example, to make Channel object whose resistance and resistive index are 3 and 1.2, user should write code like below. The first argument is a name of the channel, the second is value of the resistance, and the third is value of the resistive index.

```
Channel sampleChannel = new Channel("SampleChannel", 3, 1.2);
```

To connect two nodes with Channel object, use Connect method.

```
sampleChannel.Connect(node1, node2);
```

The first and the second arguments are Node objects to connect.

To get the energy flow in the channel, use *GetFlow* method. Figure 3.2 is a sample program which calculate energy flow between two nodes. The potential of the first and the second node are 10 and 0. The resistance and resistive index of channel are 2 and 1.2. Energy flow of this channel is $1 / 2 \times (10 - 0)^{(1/1.2)} = 3.4$.

As same as the Node class, an immutable interface (*ImmutableChannel*) for the Channel class is defined. The Channel class implements the *ImmutableChannel* interface.

```
1  /// <summary>Circuit test 1</summary>
2  /// <remarks>Calculating energy flow between two nodes</remarks>
3  private static void circuitTest1()
4  {
5      //Create new instance of Node class.
6      Node node1 = new Node("SampleNode1", 0, 10);
7      Node node2 = new Node("SampleNode2", 0, 0);
8
9      //Create new instance of Channel class and connect nodes.
10     Channel channel = new Channel("SampleChannel", 2, 1.2);
11     channel.Connect(node1, node2);
12
13     //Calculate energy flow.
14     double flow = channel.GetFlow();
15
16     Console.WriteLine("Energy flow is : " + flow.ToString("F2"));
17     Console.Read();
18 }
```

Fig.3.2 A sample program which calculate energy flow between two nodes

3.2.3 Circuit class

A circuit consists of nodes and channels. To define the circuit, Add Node objects to Circuit object and connect them with Channel objects. To add Node objects to a Circuit object, use *AddNode* method.

```
ImmutableNode iNode = sampleCircuit.AddNode(sampleNode);
```

The return value of *AddNode* method is the *ImmutableNode* object which was added to the Circuit object by *AddNode* method. To remove Node objects from a Circuit object, use *RemoveNode* method.

```
sampleCircuit.RemoveNode(iNode);
```

The first argument is the removing *ImmutableNode* object.
To connect nodes, use *ConnectNodes* method.

```
ImmutableChannel iChannel = sampleCircuit.ConnectNodes(iNode1, iNode2, sampleChannel);
```

The first and the second arguments are *ImmutableNode* objects, and the third argument is a Channel object which connects two nodes. The two nodes should be added before calling *ConnectNodes* method. The returned value of *ConnectNodes* method is the *ImmutableChannel* object which connects two nodes.
To disconnect nodes, use DisconnectNodes method.

```
sampleCircuit.DisconnectNodes(iChannel);
```

The first argument is the Channel object which connects nodes.

The boundary conditions described at section 3.2.1 could be also set in Circuit class. Table 3.2 shows lists of the method to set boundary conditions which defined in Circuit class.

Table 3.2 The method to set boundary conditions

Name	Function	Argument 1	Argument 2
SetPotential	Set initial value of potential to node	Value of the potential (double)	Target node (ImmutableNode)
SetExternalFlow	Set energy flow from node to outside	Value of the energy flow (double)	Target node (ImmutableNode)
SetBoundaryNode	Set whether node is boundary node or not.	Whether node is boundary node or not (bool)	Target node (ImmutableNode)

3.2.4 CircuitSolver class

The *CircuitSolver* class has a function to solve a circuit network. It takes a Circuit object as an argument of constructor.

```
CircuitSolver cSolver = new CircuitSolver(sampleCircuit);
```

To solve a circuit network, use Solve method.

```
cSolver.Solve();
```

All the value of energy flows and potentials are updated when the Solve method is called. There are two kinds of circuit network, circuit network who has a capacity or not. The capacity is a variable m in equation 3.1. If all the capacities in a network are equal to 0, the network is a static network. In contrast, a network whose capacities takes positive value, is a dynamic network. If a network is a static network, the value of potentials or energy flows will not change unless boundary conditions change. If a network is a dynamic network, the value of potentials or energy flows depends on time. Therefore, user should set time step when a network is a dynamic network. To set time step, use *TimeStep* property as below. In this case, time step is set to 10 seconds.

```
cSolver.TimeStep = 10;
```

3.3 Sample programs calculating circuit networks

In this section, two sample programs which calculate circuit networks are given. One is a static network which represent a water pipe network. The other is a dynamic network which represent heat flow through a wall.

3.3.1 Calculating a water pipe network

Figure 3.3 shows a water pipe network to solve[†1)]. The network has three nodes (1, 2 and 3) and four channels (A, B, C and D). The resistance of each channel are shown in figure (R_A ~ R_D).

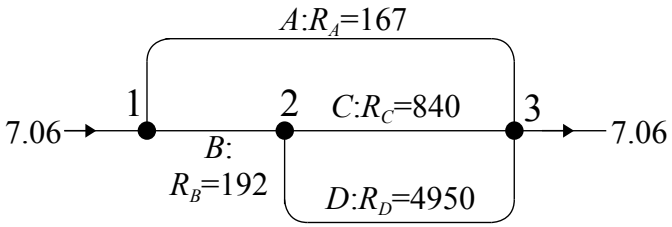


Fig.3.3 A water pipe network to solve

Figure 3.4 is a sample program which solve this network.

New instance of Circuit class is created in the line 5. Three nodes are added in the line 8~10. Since the capacities of these nodes are 0, this network is a static network. The node 1 and the node 2 is connected to outside and water flow rate is 7.06. This external water flow is set in the line 12 and 13.

The channels are created in the line 16 to 19. Since this is a water network, value of the resistive index is 2. In line 22 to 25, the nodes are connected by the channels.

In the line 29, created water flow network is solved by *CircuitSolver* object. All the potentials and water flow are calculated and written to the standard output stream. Figure 3.5 shows the result of the program.

```
1  /// <summary>Circuit test 2</summary>
2  /// <remarks>Calculating water pipe network</remarks>
3  private static void circuitTest2()
4  {
5      Circuit circuit = new Circuit("Circuit network of water pipe");
6
7      //Add nodes to circuit network
8      ImmutableNode node1 = circuit.AddNode(new Node("1", 0, 0));
9      ImmutableNode node2 = circuit.AddNode(new Node("2", 0, 0));
10     ImmutableNode node3 = circuit.AddNode(new Node("3", 0, 0));
11     //Set external water flow
12     circuit.SetExternalFlow(-7.06, node1);
13     circuit.SetExternalFlow(7.06, node3);
14
15     //Create channels
16     Channel chA = new Channel("A", 167, 2);
17     Channel chB = new Channel("B", 192, 2);
18     Channel chC = new Channel("C", 840, 2);
19     Channel chD = new Channel("D", 4950, 2);
20
21     //Connect nodes with channels
22     ImmutableChannel channelA = circuit.ConnectNodes(node1, node3, chA);
23     ImmutableChannel channelB = circuit.ConnectNodes(node1, node2, chB);
24     ImmutableChannel channelC = circuit.ConnectNodes(node2, node3, chC);
25     ImmutableChannel channelD = circuit.ConnectNodes(node2, node3, chD);
26
27     //Create solver
28     CircuitSolver cSolver = new CircuitSolver(circuit);
29     cSolver.Solve();
30     Console.WriteLine("Water flow A is " + channelA.GetFlow().ToString("F2"));
31     Console.WriteLine("Water flow B is " + channelB.GetFlow().ToString("F2"));
32     Console.WriteLine("Water flow C is " + channelC.GetFlow().ToString("F2"));
33     Console.WriteLine("Water flow D is " + channelD.GetFlow().ToString("F2"));
34     Console.Read();
35 }
```

Fig.3.4 A sample program which solve the water flow network

†1) 本問題の手計算による解法については建築設備基礎（著 木村建一）を参照して下さい。

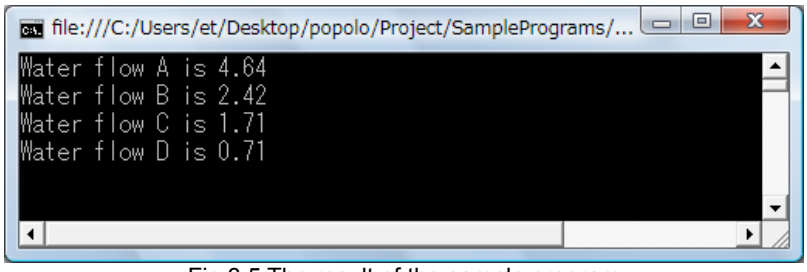


Fig.3.5 The result of the sample program

3.3.2 Calculating heat flow through a wall

Figure 3.6 is a wall to calculate a heat transfer. The wall has four layers; plywood, concrete, air gap and rock wool. The temperatures of Room 1 and Room 2 take constant value 20°C and 10°C. The temperatures of each layer are treated as potentials of node in this case.

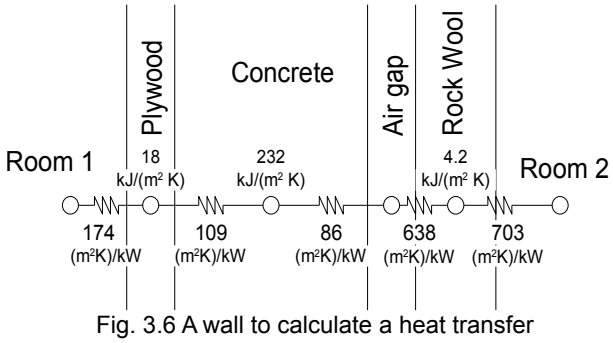


Fig. 3.6 A wall to calculate a heat transfer

Figure 3.7 is a sample program which solve the heat transfer of this wall.

In the line 8 to 14, new instances of Node class are created. In this case, each nodes has heat capacity. Since the room temperatures take constant value, edge nodes is set to boundary nodes in line 17 and 18.

As same as water pipe network, a *CircuitSolver* object is used to solve the network. In this case, time step should be set since the network has heat capacity and is a dynamic network. In the line 32, *TimeStep* is set to 3600 seconds (= 1 hour). Update method is called 24 times. The temperatures (potentials) of layers are written to the standard output stream in each iteration.

Figure 3.8 is a wall temperature distribution made from the result of the program.

```
/// <summary>Circuit test 3</summary>
/// <remarks>Calculating heat transfer through a wall</remarks>
private static void circuitTest3()
{
    Circuit circuit = new Circuit("Heat transfer network through wall");

    //Add nodes to circuit network
    ImmutableNode[] nodes = new ImmutableNode[6];
    nodes[0] = circuit.AddNode(new Node("Room 1", 0));
    nodes[1] = circuit.AddNode(new Node("Plywood", 17.9));
    nodes[2] = circuit.AddNode(new Node("Concrete", 232));
    nodes[3] = circuit.AddNode(new Node("Air gap", 0));
    nodes[4] = circuit.AddNode(new Node("Rock wool", 4.2));
    nodes[5] = circuit.AddNode(new Node("Room 2", 0));

    //Set boundary conditions (Room air temperatures).
    circuit.SetBoundaryNode(true, nodes[0]);
    circuit.SetBoundaryNode(true, nodes[5]);
    //Set air temperatures.
    circuit.SetPotential(20, nodes[0]);
    circuit.SetPotential(10, nodes[5]);
    for (int i = 1; i < 5; i++) circuit.SetPotential(10, nodes[i]); //Initialize wall temperatures to 10 C.

    //Connect nodes.
    ImmutableChannel channel01 = circuit.ConnectNodes(nodes[0], nodes[1], new Channel("Room 1-Plywood", 174, 1));
    ImmutableChannel channel12 = circuit.ConnectNodes(nodes[1], nodes[2], new Channel("Plywood-Concrete", 109, 1));
    ImmutableChannel channel34 = circuit.ConnectNodes(nodes[2], nodes[3], new Channel("Concrete-Air gap", 86, 1));
    ImmutableChannel channel45 = circuit.ConnectNodes(nodes[3], nodes[4], new Channel("Air gap-Rock wook", 638, 1));
    ImmutableChannel channel56 = circuit.ConnectNodes(nodes[4], nodes[5], new Channel("Rock wool-Room 2", 703, 1));

    CircuitSolver cSolver = new CircuitSolver(circuit);
    cSolver.TimeStep = 3600;

    for (int i = 0; i < nodes.Length; i++) Console.Write(nodes[i].Name + "  ");
    Console.WriteLine();
    for (int i = 0; i < 24; i++)
    {
        cSolver.Solve();
        Console.Write((i + 1) + "H : ");
        for (int j = 0; j < nodes.Length; j++) Console.Write(nodes[j].Potential.ToString("F1") + "  ");
        Console.WriteLine();
    }
    Console.Read();
}
```

Fig.3.7 A sample program which solve the heat transfer of the wall.

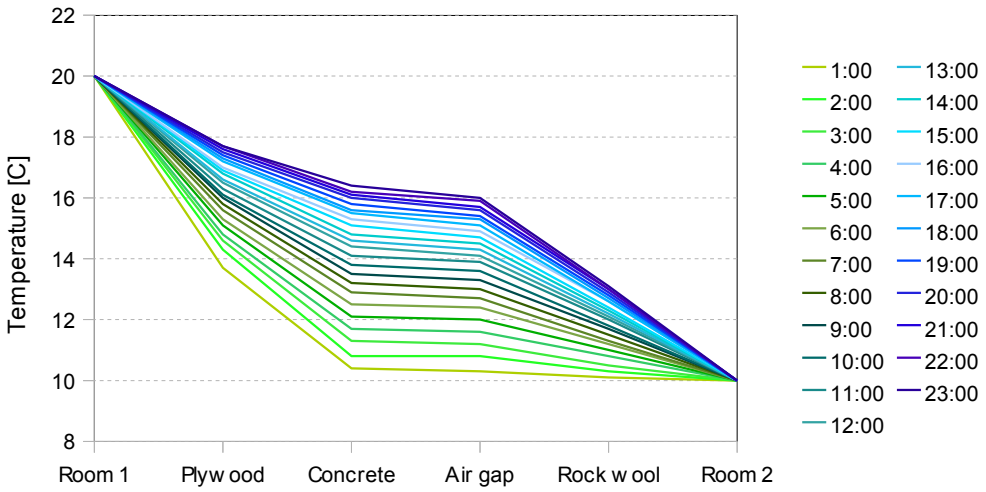


Fig.3.8 Wall temperature distribution

Chapter 4 Calculating Thermal Comfort

4.1 Name space for calculating thermal comfort

The classes which calculate thermal comfort of human belong to "*Popolo*.*ThermalComfort*" name space. Table 4.1 shows principal members defined in "*Popolo*.*ThermalComfort*" namespace.

Table 4.1 Principal members defined in "Popolo.ThermalComfort" namespace

Name	Type	General function
PMVCalculator	static class	A static class which calculate PMV value.
SETStarCalculator	static class	A static class which calculate SET* value.
HumanBody	class	A class which expresses a body of human.
BodyPart	class	A class which expresses a part of a human body.

4.2 General descriptions of classes in ThermalComfort name space

4.2.1 PMVCalculator class

PMV represents the "predicted mean vote" of a large population of people exposed to a certain environment. PPD is the predicted percent of dissatisfied people at each PMV. As PMV changes away from zero in either the positive or negative direction, PPD increases. These indices are proposed by P. O. Fanger.

To calculate value of PMV and PPD, use *PMVCalculator* class. Table 4.2 shows a list of principal members of *PMVCalculator* class.

Table 4.2 Principal members of PMVCalculator class

Name	Type	Description
Tasks	enumerator	An enumerator which expresses a kind of work
GetMet	method	Calculate metabolic rate for a task.
GetPMVFromPPD	method	Calculate PMV value from PPD value.
GetPPDFromPMV	method	Calculate PPD value from PMV value.
TryCalculateDryBulbTemperature	method	Calculate dry-bulb temperature from PMV and other thermal conditions.
TryCalculateHeatLossFromBody	method	Calculate heat loss from a human body.
TryCalculatePMV	method	Calculate PMV value from 6 thermal conditions.
TryCalculateRelativeHumidity	method	Calculate relative humidity from PMV and other thermal conditions.

1) Tasks

Tasks is an enumerator which expresses a kind of work. Some typical tasks (rest, office work, sleeping, exercise and so on) exemplified in ASHRAE Standard 55 is defined.

2) GetMet

GetMet method calculates metabolic rate for a task. It takes Tasks object as an argument.

3) GetPMVFromPPD

GetPMVFromPPD method calculates PMV value from PPD value. Since the curve of PPD is bilaterally symmetric as shown in figure 4.1, 2 PMV value corresponds to 1 PPD value. This method calculate positive value. You can get negative value by reversing a sign.

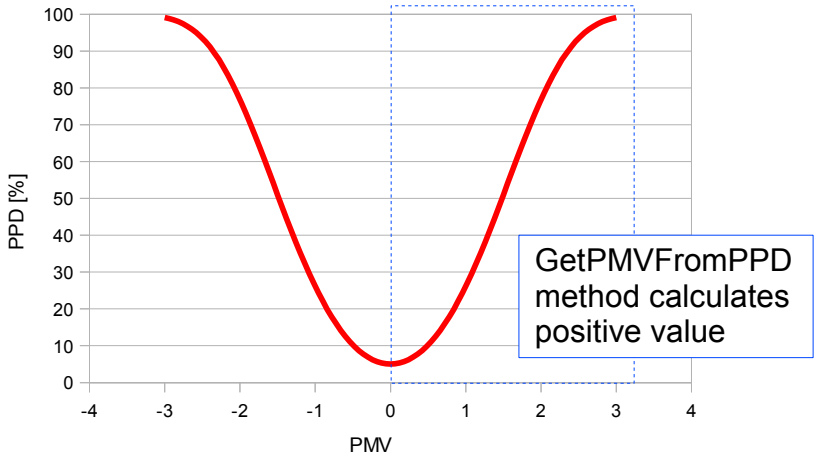


Table 4.1 PMV-PPD curve

4) GetPPDFromPMV

GetPPDFromPMV method calculate PPD value from PMV value. An argument is a value of PMV.

5) TryCalculateDryBulbTemperature

TryCalculateDryBulbTemperature method calculate dry-bulb temperature from PMV value and other thermal conditions. Table 4.3 shows a list of arguments. The arguments 1 to 7 are the inputs. Figure 4.2 shows a sample program which calculate dry-bulb temperature from PMV and other thermal conditions.

Table 4.3 The arguments of TryCalculateDryBulbTemperature method

No.	Description	No.	Description
1	PMV value [-]	5	Clo value [clo]
2	Mean radiant temperature [°C]	6	Metabolic rate [met]
3	Velocity [m/s]	7	External work [met]
4	Relative Humidity [%]	8	Dry-bulb temperature [°C] (Output)

```
1  /// <summary>PMV Test</summary>
2  private static void pmvTest()
3  {
4      double dbt;
5      PMVCalculator.TryCalculateDryBulbTemperature(1.2, 25.6, 50, 0.1, 0.8, 1.2, 0, out dbt);
6      Console.WriteLine(dbt);
7      Console.Read();
8  }
```

Fig.4.2 A sample program which calculate dry-bulb temperature from PMV and other thermal conditions

6) TryCalculateHeatLossFromBody

TryCalculateHeatLossFromBody method calculates a heat loss from a body. Table 4.4 shows a list of arguments. The arguments 1 to 7 are the inputs. The PMV value is calculated from heat loss from body.

Table 4.4 The arguments of TryCalculateHeatLossFromBody nethod

No.	Description	No.	Description
1	Dry-bulb temperature [°C]	5	Clo value [clo]
2	Mean radiant temperature [°C]	6	Metabolic rate [met]
3	Velocity [m/s]	7	External work [met]
4	Relative Humidity [%]	8	Heat loss from body [W] (Output)

7) TryCalculatePMV

TryCalculatePMV method calculates PMV value from 6 thermal conditions. Table 4.5 shows a list of arguments. The method is overloaded, and could be called with arguments listed in table 4.6.

Table 4.5 The arguments of TryCalculatePMV methods 1

No.	Description	No.	Description
1	Dry-bulb temperature [°C]	5	Clo value [clo]
2	Mean radiant temperature [°C]	6	Metabolic rate [met]
3	Velocity [m/s]	7	External work [met]
4	Relative Humidity [%]	8	PMV value [-] (Output)

Table 4.6 The arguments of TryCalculatePMV methods 2

No.	Description	No.	Description
1	Metabolic rate [met]	3	Heat loss from body [W]
2	External work [met]		

4.2.2 SETStarCalculator class

The ET* and the SET* (Standard effective temperature) are the predictive index of human response to the thermal environment proposed by Gagge[4]. To calculate the value of ET* and the value SET*, use *TryCalculateSET* method. Table 4.7 shows a list of arguments. The arguments 1 to 10 are the inputs, and the 11th and 12th arguments are the outputs.

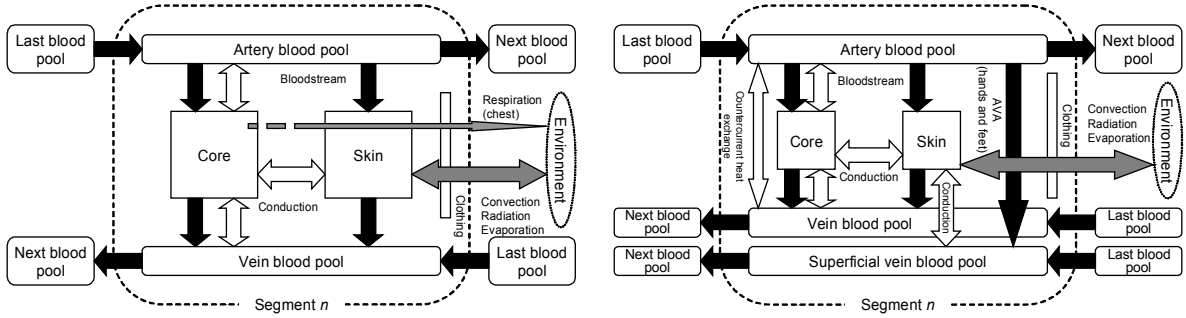
Table 4.7 The arguments of the TryCalculateSET methodz

No.	Description	No.	Description
1	Dry-bulb temperature [°C]	7	External work [met]
2	Mean radiant temperature [°C]	8	Atmospheric pressure [kPa]
3	Velocity [m/s]	9	Weight [kg]
4	Relative Humidity [%]	10	Body surface area [m2]
5	Clo value [clo]	11	ET* [-] (Output)
6	Metabolic rate [met]	12	SET* [-] (Output)

4.2.3 HumanBody class

During the past 50 years of thermal comfort research, PMV and standard new effective temperature SET* have gained support as indoor thermal comfort indices that are still the most widely utilized indices in the field today. Both indices modeled the human body as a uniform heating element and clothing as the uniform heat resistance on human body surface, limiting the application of these indices to fairly uniform condition with few distributions. However, people are often exposed to very non-uniform thermal environments, each human body segment has physiological and geometric characteristic and clothing insulation of each segment is various.

HumanBody class and *BodyPart* class are developed to calculate human body model proposed by S.Tanabe[5]. It enables detailed consideration of non-uniform conditions is required to assess such environments as car cabins, task air-conditioned spaces and outdoors. The conceptual figure of heat exchange is illustrated in Fig.4.3.



<Except limbs> <Limbs>

Figure.4.3 The conceptual figure of heat exchange

HumanBody class has two constructors. An instance of *HumanBody* is initialized to body of standard adult male. The standard man represents the body surface area of 1.87 m^2 ;the body weight of 74.43 kg; age of 25; cardiac index at rest of 2.58 L/(min m^2); fat percentage of 15 %. These parameters can be initialized when given as an arguments as below.

```
public HumanBody(double weight, double height, double age, bool isMale,
                 double cardiacIndexAtRest, double fatPercentage)
```

To specify the part of the body, *Nodes* enumerator is defined in *HumanBody* class. Table 4.8 shows members of the *Nodes* enumerator.

Table 4.8 The members of the Nodes enumerator

Name	Description	Name	Description
Head	Head	LeftLeg	Left leg
Neck	Neck	LeftFoot	Left foot
Chest	Chest	RightShoulder	Right shoulder
Back	Back	RightArm	Right arm
Pelvis	Pelvis	RightHand	Right hand
LeftShoulder	Left shoulder	RightThigh	Right thigh
LeftArm	Left arm	RightLeg	Right leg
LeftHand	Left hand	RightFoot	Right foot
LeftThigh	Left thigh	-	-

Table 4.9 shows methods defined in *HumanBody* class. A temperature of whole body can be initialized with *InitializeTemperature* method. However, the initialization of the body temperatures is not necessary, since they are initialized to steady state at operative temperature of 28.8 °C when the object is instantiated.

The state of the human body can be updated by the *Update* method. User can give the time step as an argument.

HumanBody object consists of 17 *BodyPart* objects. The each *BodyPart* object can be obtained by calling *GetBodyPart* method. *BodyPart* object has detailed information of the body (temperatures of the core, muscle, fat and skin, etc.).

Table 4.9 The methods defined in HumanBody class

Name	Description			
Update	Function	Update body state.		
	Return	-		
	Arg.1	Time step [sec]	-	-
InitializeTemperature	Function	Initialize body temperature.		
	Return	-		
	Arg.1	Temperature to initialize. [°C]	-	-
GetBodyPart	Function	Get part of human body.		
	Return	Part of the body (BodyPart object)		
	Arg.1	Part of the body (Nodes enum.)	-	-

Table 4.10 shows methods to set the boundary conditions. By giving *Nodes* enumerating object as an first argument, different boundary condition can be given to different body part (only methods which has *).

Table 4.10 Members of the HumanBody class : Methods to set boundary cnditions

Name	Description			
SetCardiacIndex	Function	Set cardiac index [L/(min m^2)]		
	Arg. 1	Cardiac index[L/(min m^2)] (double)	-	-
SetPosture	Function	Set body posture		
	Arg. 1	Body posture (BodyPosture type)	-	-
SetWorkLoad	Function	Set external work load [W/m^2]		
	Arg. 1	External work load[W/m^2] (double)	-	-
※SetContactPortionRate	Function	Set contact portion rate [-]		
	Arg. 1	Contact portion rate [-] (double)	-	-
SetHeatConductance ToMaterial	Function	Set heat conductance to external material [W/(m^2 K)]		
	Arg. 1	Body position to set heat coductance (Nodes type)	Arg. 2	Heat conductance to external material [W/(m^2 K)]
※SetDrybulbTemperature	Function	Set dry-bulb temperature [°C] of ambient		
	Arg. 1	Dry-bulb temperature [°C] (double)	-	-
※SetRelativeHumidity	Function	Set relative humidity [%] of ambient moist air		
	Arg. 1	Relative humidity [%] (double)	-	-
※SetMeanRadiant Temperature	Function	Set mean radiant temperature [°C] of ambient		
	Arg. 1	Mean radiant temperature [°C] (double)	-	-
※SetMaterialTemperature	Function	Set temperature [°C] of the contacted material		
	Arg. 1	Temperature [°C] of the contacted material (double)	-	-
※SetClothingIndex	Function	Set clothing index [clo]		
	Arg. 1	Clothing index [clo] (double)	-	-
※SetVelocity	Function	Set velocity [m/s] of ambient		
	Arg. 1	Velocity [m/s] (double)	-	-

Table 4.11 shows methods to get a state of the human body. These are the methods to get information about whole body. To get detailed information of particular body part, get *BodyPart* object by *GetBodyPart* method, and use methods defined in *BodyPart* class.

Table 4.11 Members of the HumanBody class : Methods to get state of human body

Name	Description	
GetBloodFlow	Function	Get blood flow rate [L/h] of whole human body
	Return	Blood flow rate [L/h] (double)
GetSensibleHeatLossFromSkin	Function	Get sensible heat loss [W] from whole human body skin surface
	Return	Sensible heat loss [W] from whole human body skin surface (double)
GetLatentHeatLossFromSkin	Function	Get latent heat loss [W] from whole human body skin surface
	Return	Latent heat loss [W] from whole human body skin surface (double)
GetMetabolicRate	Function	Get metabolic rate [W] of whole human body. It includes basic metabolic rate, external work and shivering
	Return	Metablic rate [W] of human body (double)
GetAverageSkinTemperature	Function	Get average skin temperature [°C] of whole human body.
	Return	Average skin temperature [°C] (double 型)

Table 4.12 shows method to get a state of body part. To specify detailed position of body part, use *Segments* enumerator. Table 4.13 shows members of *Segment* enumerator.

Table 4.12 The methods defined in BodyPart class

Name	Description			
GetSensibleHeatLoss	Description	Calculate sensible heat loss [W] from the skin.		
	Return	Sensible heat loss [W] from the skin. (double)		
	-	-	-	-
GetHeatCapacity	Description	Get segment heat capacity [Wh/K] of the body part		
	Return	Segment heat capacity [W/K] of the body part. (double)		
	Arg.1	Segment of the body part (Segments object)	-	-
GetTemperature	Description	Get segment temperature [°C] of the body part		
	Return	Segment temperature [°C] of the body part (double)		
	Arg.1	Segment of the body part (Segments object)	-	-
GetHeatConductance	Description	Get heat conductance [W/K] between the segments. If the segments are not contacted, 0 is returned.		
	Return	Heat conductance [W/K] between the segments.		
	Arg.1	Segment 1 (Segments object)	Arg.2	Segment 2 (Segments object)
GetMetabolicRate	Description	Get segment metabolic rate [W] of the body part		
	Return	Segment metabolic rate [W] of the body part (double)		
	Arg.1	Segment (Segments object)	-	-
GetBloodFlow	Description	Get blood flow [L/h] at segment of the body part		
	Return	Blood flow [L/h] segment (double)		
	Arg.1	Segment (Segments object)	-	-
GetHeatTransfer	Description	Calculate heat transfer [W] from a segment 1 to a segment 2.		
	Return	Heat transfer[W] from a segment 1 to a segment 2 (double)		
	Arg.1	Segment 1 (Segments object)	Arg.2	Segment 2 (Segments object)

Table 4.13 The members of Segments enumerator

Name	Description	Name	Description
Core	Core layer	Artery	Artery
Muscle	Muscle layer	SuperficialVein	Superficial vein
Fat	Fat layer	DeepVein	Deep vein
Skin	Skin layer	AVA	AVA blood

Figure 4.4 shows a sample program which calculates thermal state of human body.

In the line 8, *HumanBody* object is initialized. It is the model of female and the weight, height, age and cardiac index are 70kg, 1.6m, 35, 2.58 L/(min m^2) respectively.

In the line 10 to 19, the boundary conditions are given. This is the code to set boundary conditions simultaneously to whole body parts. To set boundary conditions individually to each body part, give *BodyParts* object as a first argument as line 22. In this case, dry-bulb temperature just around right hand settled at 20 °C.

The state of body is updated in the line 28. The time step is 120 seconds. The elapse of the time is 30 minutes, since the *Update* method is called 16 times. Of course, the boundary conditions can be changed at each iteration.

In the line 29 and 30, the left and the right shoulder objects are taken out to get information. The temperatures of the core layer and the skin layer is calculated. They are written to the standard output stream.

Figure 4.5 shows the result of the simulation. The body temperature became higher as the time past. Since the dry-bulb temperature around the right hand settled at 20°C, the temperature of the right shoulder was lower than that of the left shoulder.

```
1  /// <summary>Sample program calculating human body</summary>
2  private static void humanBodyTest()
3  {
4   //This is constructor to make standard human body.
5   //HumanBody body = new HumanBody();
6
7   //Make human body model : Weight 70kg, Height 1.6m, Age 35, Female, Cardiac index 2.58, Fat 20%
8   HumanBody body = new HumanBody(70, 1.6, 35, false, 2.58, 20);
9
10  //Set clothing index [clo]
11  body.SetClothingIndex(0);
12  //Set dry-bulb temperature [C]
13  body.SetDrybulbTemperature(42);
14  //Set mean radiant temperature [C]
15  body.SetMeanRadiantTemperature(42);
16  //Set velocity [m/s]
17  body.SetVelocity(1.0);
18  //Set relative humidity [%]
19  body.SetRelativeHumidity(50);
20
21  //Use Nodes enumarator to set bouncary condition to particular position
22  body.SetDrybulbTemperature(HumanBody.Nodes.RightHand, 20);
23
24  //Updating body state
25  Console.WriteLine("Time     | R.Shoulder C temp | R.Shoulder S temp | L.Shoulder C temp | L.Shoulder S temp");
26  for (int i = 0; i < 15; i++)
27  {
28   body.Update(120);
29   ImmutableBodyPart rightShoulder = body.GetBodyPart(HumanBody.Nodes.RightShoulder);
30   ImmutableBodyPart leftShoulder = body.GetBodyPart(HumanBody.Nodes.LeftShoulder);
31   Console.Write(((i + 1)  * 120) + "sec | ");
32   Console.Write(rightShoulder.GetTemperature(BodyPart.Segments.Core).ToString("F2") + " | ");
33   Console.Write(rightShoulder.GetTemperature(BodyPart.Segments.Skin).ToString("F2") + " | ");
34   Console.Write(leftShoulder.GetTemperature(BodyPart.Segments.Core).ToString("F2") + " | ");
35   Console.Write(leftShoulder.GetTemperature(BodyPart.Segments.Skin).ToString("F2") + " | ");
36   Console.WriteLine();
37  }
38
39  Console.Read();
40 }
```

Figure 4.4 The sample program which calculates thermal state of human body

```
Time     | R.Shoulder C temp | R.Shoulder S temp | L.Shoulder C temp | L.Shoulder S temp
120sec   | 35.89 | 35.70 | 35.89 | 35.74 |
240sec   | 35.91 | 36.07 | 35.92 | 36.17 |
360sec   | 35.93 | 36.20 | 35.96 | 36.35 |
480sec   | 35.96 | 36.21 | 36.02 | 36.38 |
600sec   | 35.99 | 36.17 | 36.07 | 36.36 |
720sec   | 36.02 | 36.13 | 36.12 | 36.32 |
840sec   | 36.05 | 36.10 | 36.17 | 36.28 |
960sec   | 36.08 | 36.08 | 36.22 | 36.26 |
1080sec | 36.11 | 36.06 | 36.27 | 36.24 |
1200sec | 36.14 | 36.05 | 36.31 | 36.22 |
1320sec | 36.17 | 36.04 | 36.35 | 36.21 |
1440sec | 36.19 | 36.03 | 36.38 | 36.20 |
1560sec | 36.21 | 36.03 | 36.41 | 36.20 |
1680sec | 36.24 | 36.03 | 36.44 | 36.19 |
1800sec | 36.26 | 36.02 | 36.47 | 36.19 |
```

Figure 4.5 The result of the simulation

Chapter 5 Calculating Weather State

5.1 Name space for calculating weather state

Classes which calculate weather state belong to "*Popolo.Weather*" name space. Table 5.1 shows principal members defined in "*Popolo.Weather*" name space.

Table 5.1 Principal members defined in Popolo.Weather name space

Name	Description
Incline	A class which expresses an incline.
Sky	A static class which calculates various weather phenomena.
Sun	A class which expresses sun.
WeatherData	A class which contains weather data.
WeatherDataTable	A class which contains weather data. It consists of WeatherRecord objects.
WeatherRecord	A class which contains weather data. It consists of WeatherData objects.

5.2 Descriptions of classes

5.2.1 Incline class

To calculates nocturnal radiation or radiation from sun, 3 dimensional positioning between plane and point should be clearly defined. An incline can be defined with horizontal vertical angle. The *Incline* class takes these 2 value as an arguments of constructor. The constructor of Incline is overloaded, and there are 3 kinds of arguments as shown in Table 5.2.

Table 5.2 The arguments of Incline constructors

No.	Argument 1	Argument 2
1	Horizontal angle : Unit is radian. South is 0, west is negative, east is positive. (double)	Vertical angle : Unit is radian. Horizontal surface is 0, Vertical surface is 1/2π (double)
2	Orientation (Orientation object)	Vertical angle : Unit is radian. Horizontal surface is 0, Vertical surface is 1/2π (double)
3	Incline to copy (ImmutableIncline object)	-

The first argument of the second constructor is *Orientation* object which is defined in *Incline* class. It can express 16 orientations (N, NNW, NW, WNW, W, WSW, SW, SSW, S, SSE, SE, ESE, E, ENE, NE, NNE).

Vertical angle of incline is expressed by radian, where horizontal surface is 0 and vertical surface is 1/2π.

The third constructor is a copy constructor. With calling third constructor, *Incline* object is initialized with the given *ImmutableIncline* object.

Table 5.3 and table 5.4 shows the methods and properties defined in the *Incline* class.

Table 5.3 The properties defined in the Inline class

Name	Description	Has set accessor	Unit	Type
ConfigurationFactorToSky	Configuration factor to the sky	-	-	double
HorizontalAngle	Horizontal angle of the inclined plane	-	Radian South is 0, west is negative, east is positive	double
VerticalAngle	Vertical angle of the inclined plane	-	Radian Horizontal surface is 0, vertical surface is 1/2π	double

Table 5.4 The methods defined in the Incline class

Name	Description			
GetDirectSolarRadiationRate	Description	Calculate cosine θ of incident angle to the normal direction of the inclined plane.		
	Return	Cosine θ of incident angle to the normal direction of the inclined plane.		
	Arg.1	Sun (ImmutableSun object)	-	-
GetTanPhiAndGamma	Description	Calculate profile angle φ and angle between solar azimuth and normal direction of the inclined plane.		
	Return	-		
	Arg.1	Sun (ImmutableSun object)	Arg.2	Output : Tangent of profile angle φ
	Arg.3	Output :Tangent of angle between solar altitude and normal direction of the inclined plane		
Reverse	Description	Reverse the Incline object		
	Return	-		

To calculate a radiation from sun, 3D positional relation between sun and incline (Figure 5.1).

γ is an angle between solar azimuth and normal direction of the inclined plane. φ is an angle between solar altitude and normal direction of the inclined plane. φ is called profile angle. φ and γ can be obtained by *GetTanPhiAndGamma* method. The first argument is a *Sun* object, which will be described later in section 5.2.3. The 2nd and 3rd argument are the outputs, tangent φ and tangent γ.

θ is an angle between incident angle and the normal direction of the inclined plane. θ can be obtained by *GetDirectSolarRadiationRate* method. The first argument is a *Sun* object. The returned value is cosine θ. This value is used when calculate an insolation on the inclined plane from the direct normal radiation.

The inclined plane can be reversed with calling *Reverse* method.

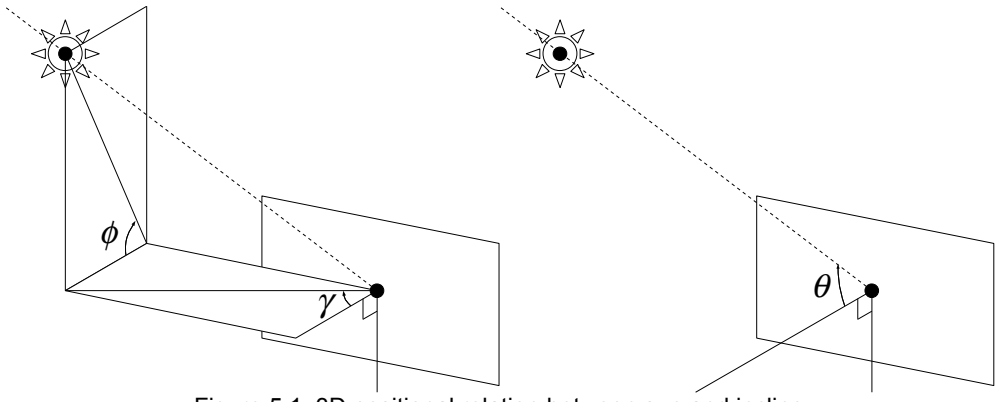


Figure 5.1 3D positional relation between sun and incline

5.2.2 Sky class

Various static methods related to sky state are defined in the *Sky* class. Table 5.5 shows the methods defined in the *Sky* class.

Table 5.5 The methods defined in the Sky class

Name	Description			
DegreeToRadian	Description	Convert an unit of angle from degree to radian		
	Return	Angle [radian]		
	Arg.1	Angle [degree]	-	-
GetAtmosphericRadiation	Description	Calculate atmospheric radiation [W/m^2]		
	Return	Atmospheric radiation [W/m^2]		
	Arg.1	Dry-bulb temperature of outdoor moist air[C]	Arg.2	Cloud cover ratio [-] (0~1)
	Arg.3	Water vapor pressure [kPa]		
GetNocturnalRadiation	Description	Calculate nocturnal radiation [W/m^2]		
	Return	Calculate nocturnal radiation [W/m^2]		
	Arg.1	Dry-bulb temperature of outdoor moist air[C]	Arg.2	Cloud cover ratio [-] (0~1)
	Arg.3	Water vapor pressure [kPa]		
RadianToDegree	Description	Convert an unit of angle from radian to degree.		
	Return	Angle [degree]		
	Arg.1	Angle [radian]	-	-

5.2.3 Sun class

1) Constructor

The *Sun* class expresses sun. Information about longitude and latitude at the location must be given as argument in the constructor of the Sun to calculate azimuth and altitude of sun. Table 5.6 shows the constructors of the *Sun* class.

Table 5.6 The arguments of the Sun class

No.	Arg.1	Arg.2	Arg.3
1	Latitude [degree] (double)	Longitude [degree] (double)	Longitude of the standard time meridian [degree]
2	City to calculate sun position (City object)	-	-
3	Copy constructor (ImmutableSun)	-	-

An argument of the second constructor is *City* enumerator defined in the *Sun* class. More than 100 cities in the world are defined. A *Sun* object is initialized with longitude and latitude where the city is located. Below is a sample constructor call.

```
Sun sun = new Sun(Sun.City.Tokyo);
```

2) Static methods

Some basic methods related to sun position or solar radiation are defined in *Sun* class. Table 5.7 and Table 5.8 shows the static methods defined in the *Sun* class.

Table 5.7 The static methods related to sun position

Name	Description			
GetEquationOfTime	Description	Caluculate an equation of the time.		
	Return	Equation of the time.		
	Arg.1	Date and time (DateTime object)		
GetHourAngle	Description	Calculate hour angle from the location and the equation of the time.		
	Return	Hour angle [°]		
	Arg.1	Equation of the time.	Arg.2	Longitude [degree]
	Arg.3	Longitude of the standard time meridian [degree]	Arg.4	Date and time (DateTime object)
GetSunAltitude	Description	Calculate an altitude of sun. [radian]		
	Return	Altitude of sun. [radian]		
	Arg.1	Latitude [degree]	Arg.2	Longitude [degree]
	Args3	Longitude of the standard time meridian [degree]	Arg.4	Date and time (DateTime object)
GetSunAzimuth	Description	Caluculate an azimth of sun [radian]		
	Return	Azimth of sun. [radian]		
	Arg.1	Latitude [degree]	Arg.2	Longitude [degree]
	Arg.3	Longitude of the standard time meridian [degree]	Arg.4	Date and time (DateTime object)
GetSunDeclination	Description	Caluculate declination of sun. [degree]		
	Return	Declination of sun. [degree]		
	Arg.1	Date and time (DateTime object)	-	-
.GetSunPosition	Description	Calculate an altitude [radian] and a direction [radian] of sun		
	Return	-		
	Arg.1	Latitude [degree]	Arg.2	Longitude [degree]
	Arg.3	Longitude of the standard time meridian [degree]	Arg.4	Date and time (DateTime object)
	Arg.5	Output : Altitude of sun [radian]	Arg.6	Output : Direction of sun [radian]
GetSunRiseTime	Description	Calculate time of sun rise.		
	Return	Time of sun rise (DateTime object)		
	Arg.1	Latitude [degree]	Arg.2	Longitude [degree]
	Arg.3	Longitude of the standard time meridian [degree]	Arg.4	Date and time (DateTime object)
GetSunSetTime	Description	Calculate time of sun set.		
	Return	Time of sun set (DateTime object)		
	Arg.1	Latitude [degree]	Arg.2	Longitude [degree]
	Arg.3	Longitude of the standard time meridian [degree]	Arg.4	Date and time (DateTime object)

Table 5.8 The static methods related to solar radiation

Name	Description			
GetExtraterrestrial Radiation	Description	Calculate extraterrestrial radiation [W/m^2]		
	Return	Extraterrestrial radiation [W/m^2]		
	Arg.1	Days of the year (int object)	-	-
GetDirectNormalRadiation	Description	Calculate direct normal radiation from global horizontal radiation [W/m^2] and diffuse horizontal radiation [W/m^2]		
	Return	Direct normal radiation [W/m^2]		
	Arg.1	Global horizontal radiation [W/m^2]	Arg.2	Diffuse horizontal radiation [W/m^2]
	Arg.3	Altitude of sun [radian]	-	-
GetDiffuseHorizontal Radiation	Description	Calculate diffuse horizontal radiation [W/m^2] from diract normal radiation [W/m^2] and global horizontal radiation [W/m^2].		
	Return	Diffuse horizontal radiation [W/m^2]		
	Arg.1	Direct normal radiation [W/m^2]	Arg.2	Diffuse horizontal radiation [W/m^2]
	Arg.3	Altitude of sun [radian]	-	-
GetGlobalHorizontal Radiation	Description	Calculate global horizontal radiation from diffuse horizontal radiation [W/m^2] and direct normal radiation [W/m^2]		
	Return	Global horizontal radiation [W/m^2]		
	Arg.1	Diffuse horizontal radiation [W/m^2]	Arg.2	Direct normal radiation [W/m^2]
	Arg.3	Altitude of sun [radian]	-	-
EstimateDiffuseAnd DirectNormalRadiation	Description	Estimate diffuse horizontal radiation and direct normal radiation from global horizontal radiation.		
	Return	-		
	Arg.1	Global horizontal radiation [W/m^2]	Arg.2	Longitude [degree]
	Arg.3	Latitude [degree]	Arg.4	Longitude of the standard time meridian [degree]
	Arg.5	Date and time (DateTime object)	Arg.6	Estimating Method
	Arg.7	Output : Direct normal radiation [W/m^2]	Arg.8	Output : Diffuse horizontal radiation [W/m^2]

The value of latitude and longitude are given by degree unit (not radian). East longitude is positive value and west longitude is negative. North latitude is positive value and south latitude is negative. For example, the arguments for Tokyo city (lies at 139.45° east in longitude and 35.4° north in latitude) are 139 + 45/60 = 139.75 and 35 + 4 / 60 = 35.07. The arguments for Rio de Janeiro (lies at 22.57° west in longitude and 43.12° south in latitude) are -22 - 57/60 = -22.95 and -43 - 12/60 = -43.2.

In most cases, a climate observatory measures only the global horizontal radiation, and the direct normal radiation and the diffuse horizontal radiation are not measured. Therefore, many people proposed methods to estimate the direct normal radiation and the diffuse horizontal radiation from the global horizontal radiation[6) 7) 8) 9)]. In the *Sun* class, *EstimateDiffuseAndDirectNormalRadiation* method is defined. User can select estimating method by setting *DiffuseAndDirectNormalRadiationEstimatingMethod* enumerator for the 6th argument. 6 methods, the method of Berlage[10)], Matsuo[11)], Nagata[12)], Liu & Jordan[13)], Udagawa & Kimura[14)], Watanabe[15)], Akasaka[16)] and Miki[17)], can be selected.

3) Other properties and methods defined in Sun class

Table 5.9 shows the properties defined in the *Sun* class. Value of the properties except radiations (*DiffuseHorizontalRadiation*, *DirectNormalRadiation* and *GlobalHorizontalRadiation*) can be determined when the location and the date and time information is given. The location is given as an argument of the constructor. The date and time information can be given by *Update* method. The *DateTime* object is given as an argument.

```
sun.Update(dateTime);
```

Table 5.9 The properties defined in Sun class

Name	Description	Has set accessor	Unit	Type
Altitude	Altitude of sun	TRUE	radian	double
CurrentDateTime	Current date and time	-	-	DateTime
DiffuseHorizontalRadiation	Diffuse horizontal radiation	TRUE	W/m^2	double
DirectNormalRadiation	Direct normal radiation	TRUE	W/m^2	double
GlobalHorizontalRadiation	Global horizontal radiation	TRUE	W/m^2	double
Latitude	Latitude of sun	-	degree	double
Longitude	Longitude of sun	-	degree	double
Orientation	Orientation of sun	-	radian	double
Revision	Revision of Sun instance. Revision number is updated whenever Update method is called.	-	-	uint
StandardLongitude	Longitude of the standard time meridian	-	degree	double
SunRiseTime	Time of sun rise	-	-	DateTime
SunSetTime	Time of sun set	-	-	DateTime

Three values of radiation, the global horizontal radiation, the diffuse horizontal radiation and the direct normal radiation are related to each other. The value of one can be estimated when the rest of two value are given (Figure 5.2). Table 5.10 shows static methods to calculate these radiations.

Table 5.10 The methods to calculate a value of radiations

Name	Arg.1	Arg.2
SetDirectNormalRadiation	Global horizontal radiation	Diffuse horizontal radiation
SetDiffuseHorizontalRadiation	Direct normal radiation	Global horizontal radiation
SetGlobalHorizontalRadiation	Diffuse horizontal radiation	Direct normal radiation

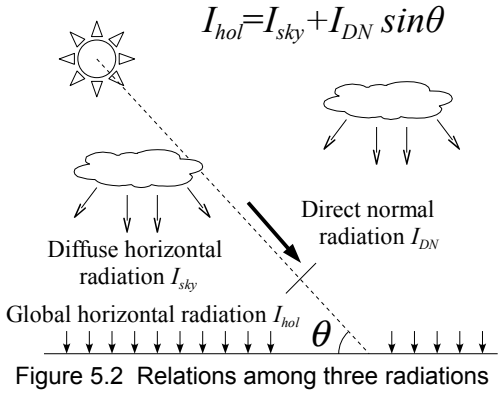


Figure 5.2 Relations among three radiations

5.3 Sample programs of calculating weather state

Figure 5.3 shows a sample program which calculate radiation of sun. It estimates the diffuse horizontal radiation and the direct normal radiation from the global horizontal radiation. The insolations on the inclined planes are also calculated. Figure 5.4 shows the result of the program.

```
/// <summary>Sample program calculating weather state</summary>
private static void weatherTest()
{
    //Create an instance of the Sun class. (Location is Tokyo)
    Sun sun = new Sun(Sun.City.Tokyo);

    //Set date and time information(1983/12/21 12:00)
    DateTime dTime = new DateTime(1983, 12, 21, 12, 0, 0);
    sun.Update(dTime);

    //Create instances of the Incline class. Vertical south east surface and 45 degree west surface.
    Incline seInc = new Incline(Incline.Orientation.SE, 0.5 * Math.PI);
    Incline wInc = new Incline(Incline.Orientation.W, 0.25 * Math.PI);

    //Estimate direct normal and diffuse horizontal radiation from global horizontal radiation (467 W/m2)
    sun.EstimateDiffuseAndDirectNormalRadiation(467);

    //Calculate insolation rate on the inclined plane.
    double cosThetaSE, cosThetaW;
    cosThetaSE = seInc.GetDirectSolarRadiationRate(sun);
    cosThetaW = wInc.GetDirectSolarRadiationRate(sun);

    Console.WriteLine("Location:Tokyo, Date and time:12/21 12:00");
    Console.WriteLine("Altitude of sun=" + Sky.RadianToDegree(sun.Altitude).ToString("F1") + " degree");
    Console.WriteLine("Orientation of sun=" + Sky.RadianToDegree(sun.Orientation).ToString("F1") + " degree");
    Console.WriteLine("Direct normal radiation=" + sun.DirectNormalRadiation.ToString("F1") + " W/m2");
    Console.WriteLine("Diffuse horizontal radiation=" + sun.GlobalHorizontalRadiation.ToString("F1") + " W/m2");
    Console.WriteLine("Direct normal radiation to SE surface=" + (sun.DirectNormalRadiation * cosThetaSE).ToString("F1") + " W/m2");
    Console.WriteLine("Direct normal radiation to W surface=" + (sun.DirectNormalRadiation * cosThetaW).ToString("F1") + " W/m2");

    Console.Read();
}
```

Figure 5.3 The sample program which calculate radiation of sun

```
Location:Tokyo, Date and time:12/21 12:00
Altitude of sun=30.7 degree
Orientation of sun=5.4 degree
Direct normal radiation=650.2 W/m2
Diffuse horizontal radiation=467.0 W/m2
Direct normal radiation to SE surface=356.1 W/m2
Direct normal radiation to W surface=272.1 W/m2
```

Figure 5.4 The result of the program

Chapter 6 Calculating Thermal Load of Building

6.1 Name space for calculating thermal load of building

Classes which calculate building thermal load belong to the "*Popolo*.T*hermalLoad*" name space. Table 6.1 shows principal members defined in the "*Popolo*.*ThermalLoad*" name space. Some classes described at the previous chapters are also used to calculate thermal load (*Sun* class, *Incline* class, *MoistAir* class and so on). Figure 6.1 shows a UML diagram which shows the relations among classes used to calculate building thermal load. Most of the classes belongs to the "*Popolo.ThermalLoad*" name space.

Table 6.1 The principal members defined in Popolo.ThemalLoad name space

Name	Description
GlassPanes	The GlassPanes class is a model of one or more glass layers.
IHeatGain	The interface which shows a heat gain in the room.
MultiRoom	The MultiRoom class is a model of a multi-room. A multi-room consists of more than two rooms that are sepalated by a wall.
Outdoor	The Outdoor class is a model of an outdoor (It includes an information about sun position, moist air state, etc.).
Room	The Room class is a model of a single room. A room consists of more than two zones, that are not sepalated by a wall (ex. Interior zone and perimetor zone).
SunShade	The SunShade class is a model of a sun shade. Only simple shaped sun shade (Horizontal, vertical or gird) could be modeled with this class.
Wall	The Wall class is a model of a multi layered wall.
WallLayers	The WallLayers class is a model of a composition of the multi layered wall.
WallMaterial	The WallMaterial class is a model of a material used for wall.
WallSurface	The WallSurface class is a model of a wall surface. It contains information about emissivity and inclined degree of wall surface.
Window	The Window class is a model of a window.
WindowSurface	The WindowSurface class is a model of window surface. It contains information about emissivity and inclined degree of window surface.
Zone	The Zone class is a model of a single zone. A zone consists of wall surfaces and window surfaces.

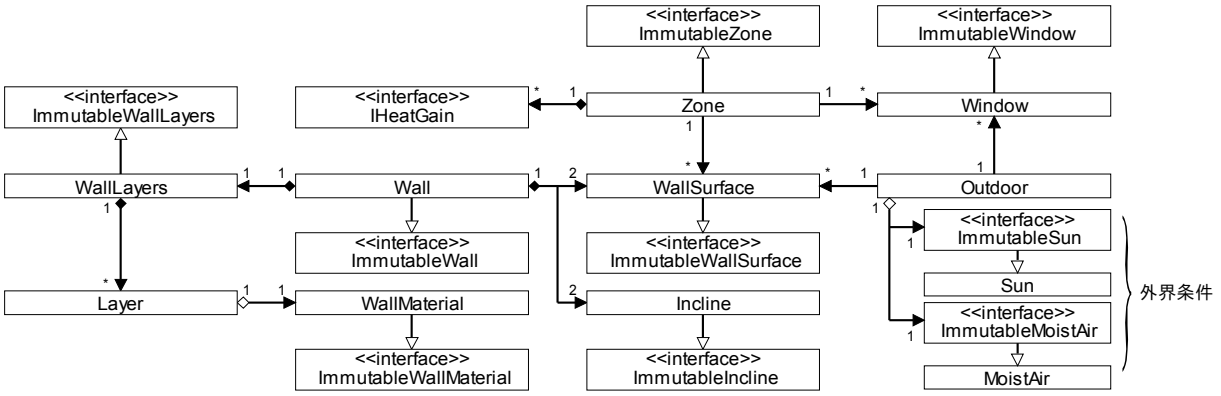


Figure 6.1 The UML diagram of classes relating to Popolo.ThermalLoad name space

6.2 General descriptions of classes defined in Popolo.ThermalLoad name space

6.2.1 GlassPanes class

GlassPanes class expresses one or more glass layers. It calculates a characteristics of a window.

Two kinds of the constructor is defined in *GlassPanes* class. Table 6.2 shows the constructors of the *GlassPanes* class. The first constructor is the simplified one, which directly set the information about window characteristics (Overall transmissivity, overall absorptivity and heat transfer coefficient). The resistances at surfaces are not involved in heat transfer coefficient (arg. 3).

The second constructor is the detailed option, which calculates inter-reflection of glass panes. In this case, the lists of the *GlassPanes.Pane* class is given as an argument.

Table 6.2 The constructors of the GlassPanes class

No.	Argument 1	Argument 2	Argument 3
1	Overall transmissivity [-] (double type)	Overall absorptivity [-] (double type)	Thermal transmittance of the glass [W/(m^2 K)] (double type)
2	Lists of the glass panes (Array of the GlassPanes.Pane objects)	-	-

There are three types of heat gains through the window of a building. The first is the transmission heat gain, the second is the absorption heat gain, and the last is the transfer heat gain (Figure 6.2). The solar incident falling on the window is partly reflected, partly absorbed and partly transmitted by the window glass. The absorbed heat partly go back to outdoor, and the rest infiltrate into the building. We call this infiltration rate to the solar incident as “overall absorptivity”.

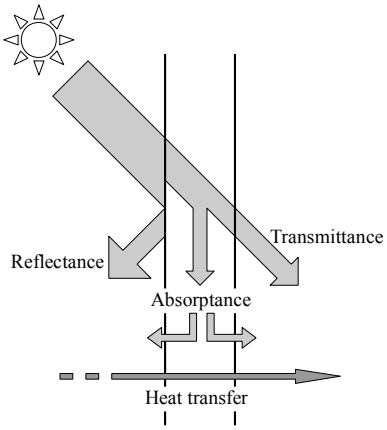


Figure 6.2 Heat gains through a window

If a window consists of a single glass pane, overall transmissivity and overall absorptivity is equal to the transmissivity and the absorptivity of the single glass pane. If a window consists of more than one glass pane, detailed calculation is needed, since there occurs inter reflections among the glass panes. To calculate this phenomenon, use the second constructor of the *GlassPanes* class. The *GlassPanes.Pane* class is a model of a single glass pane which is the inner class of the *GlassPanes* class. Table 6.3 shows constructors of the *GlassPanes.Pane* class.

Table 6.3 The constructors of the GlassPanes.Pane

No.	Argument 1	Argument 2	Argument 3	Argument 4	Argument 5
1	Transmissivity [-] (double type)	Absorptivity [-] (double type)	Heat transfer coefficient [W/m2] (double type)	-	-
2	Transmissivity of outer side [-] (double type)	Absorptivity of outer side [-] (double type)	Transmissivity of inner side [-] (double type)	Absorptivity of inner side [-] (double type)	Heat transfer coefficient [W/m2] (double type)
3	Predefined glass type. (PredifinedGlassPane type)	-	-	-	-

In some glass panes, transmissivity and absorptivity of a surface is different from that of the opposite side surface. In this case, the second constructor should be used.

Some commonly-used glass panes are already defined in *GlassPanes.Pane* class. To make these glass panes, use the third constructor. The *PredifinedGlassPane* enumerator type is given as an argument.

By giving a *GlassPanes.Pane* array (The number of elements is *M*) to *GlassPanes* constructor, Overall transmissivity and overall reflectivity of the *M* layered glass panes are calculated according to equation 6.1 to 6.5. Where $\tau_{T(M)}$ is overall transmissivity, $\rho_{T(M)}$ is overall reflectivity and $\alpha_{T(M)n}$ is absorptivity of n^{th} layer. The first element of array is the inside glass pane and the last element of array is the outside glass pane.

$$\tau_{T(m)} = \tau_{T(m-1)} X_r \tag{6.1}$$

$$\rho_{T(m)} = \rho_{Lm} + \tau_{Rm} \rho_{T(m-1)} X_r \tag{6.2}$$

$$\alpha_{T(m)n} = \alpha_{T(m-1)n} X_r \tag{6.3}$$

$$\alpha_{T(m)n} = \alpha_{Lm} + \alpha_{Lm} + \alpha_{Rm} \rho_{T(m-1)} X_r \tag{6.4}$$

$$X_r = \frac{\tau_{Lm}}{1 - \rho_{Rm} \rho_{T(m-1)}} \quad , \quad \tau_{T(1)} = \tau_{L1} \quad , \quad \rho_{T(1)} = \rho_{L1} \quad , \quad \alpha_{T(1)1} = 1 - \tau_{L1} - \rho_{L1} \tag{6.5}$$

τ_T : transmissivity [-], ρ_T : reflectivity [-], $\alpha_{T(m)n}$: absorptivity
M : number of layers, L : outer side of glass pane, R : inner side of glass pane

The thermal transmittance and the overall absorptivity of the glass panes can be calculated by equation 6.6 to 6.8.

$$K_{GS} = 1 / \left(\frac{1}{\alpha_i} + \sum_{m=0}^{M-1} \left\{ R_{a(m)} + R_{g(m)} \right\} + \frac{1}{\alpha_o} \right) \tag{6.6}$$

$$B = \sum_{m=1}^{M} f_m \alpha_{T(m)} \tag{6.7}$$

$$f_m = 1 - K_{GS} \left(\frac{1}{\alpha_i} + \sum_{n=0}^{m-1} \left\{ R_{a(n)} + R_{g(n)} \right\} \right) \tag{6.8}$$

K_{GS} : Thermal transmittance of the glass panes [W/(m^2 K)], R_a : resistance of the air gap [(m^2 K)/W]
R_h : resistance of the glass pane [(m^2 K)/W], f_m : infiltration rate of absorbed heat gain at mth layer into a room
α_o : film coefficient at outside glass surface [W/(m^2 K)], α_i : film coefficient at inside glass surface [W/(m^2 K)]
B : overall heat absorptivity of glass panes [-]

Table 6.4 shows properties of *GlassPanes* class.

Table 6.4 The properties of the GlassPanes class

Name	Description	Has set accessor	Unit	Type
AngularDependenceCoefficients	Angular dependence of the solar heat gain coefficient	TRUE	-	double[]
ConvectiveRate	Convective rate of heat gain	TRUE	-	double
ThermalTransmittanceOfGlass	Thermal transmittance of the glass (The film coefficient at surfaces are not involved).	-	W/(m^{2}K)	double
LongWaveEmissivity	Long wave emissivity of outer surface	TRUE	-	double
OverallAbsorptivity	Overall absorptivity	-	-	double
ThermalTransmittance	Thermal transmittance of the glass	-	W/(m^{2}K)	double
OverallTransmissivity	Overall transmissivity	-	-	double
RadiativeRate	Rdiative rate of heat gain	TRUE	-	double

AngularDependenceCoefficients is coefficients to calculate angular dependence of the solar heat gain. The angular dependence can be calculated by equation 6.9. Default value of coefficients are A_i = {3.4167, -4.389, 2.4948, -0.5224}. Figure 6.3 shows angular dependence with default value. These coefficients give reasonable result in the most cases.

$$\frac{\tau_D}{\tau_N} = \sum_{i=1}^{N} A_i \cos^i \theta \tag{6.9}$$

τ_N : transmissivity when the solar incident angle is 90 degree [-], τ_D : transmissivity when the solar incident angle is θ degree [-]
A_i : coefficients to calculate angular dependence of the solar heat gain [-]

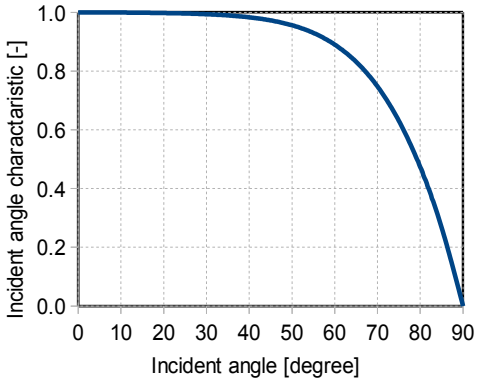


Figure 6.3 Angular dependence

ConvectiveRate and *RadiativeRate* properties shows the convective and the radiative rate of a heat gain. These properties takes value between 0 to 1.

LongWaveEmissivity is used to calculate a heat loss by a nocturnal radiation.

Film coefficients at surfaces are not involved in *HeatTransferCoefficientOfGlass*.

Table 6.5 shows methods defined in *GlassPanes* class.

Table 6.5 The methods defined in the GlassPanes class

Name	Description			
Copy	Description	Copy other GlassPanes object		
	Return	-		
	Arg.1	Target GlassPanes object to copy (ImmutableGlassPanes type)	-	-
GetIncidentAngle Characteristic	Description	Calculate incident angle characteristic		
	Return	Incident angle characteristic of the glass panes.		
	Arg.1	Cosine of incident angle (cosθ)	-	-
SetHeatTransfer CoefficientsOfAirGaps	Description	Set heat transfer coefficients [W/(m^2·K)] of air gaps		
	Return	-		
	Arg.1	Index of air gap :0 is room side	Arg.2	heat transfer coefficients [W/(m^2·K)] of air gap
SetInsideFilmCoefficient	Description	Set inside film coefficient [W/(m^2·K)].		
	Return	-		
	Arg.1	inside film coefficient [W/(m^2·K)]	-	-
SetOutsideFilmCoefficient	Description	Set outside film coefficient [W/(m^2·K)].		
	Return	-		
	Arg.1	outside film coefficient [W/(m^2·K)]	-	-

If glass panes consist of more than one pane, heat transfer coefficients of air gaps must be given by *SetHeatTransferCoefficientsOfAirGaps* method.

Figure 6.4 shows a sample program which calculates the characteristics of a glass panes. Triple layered glass panes are created and some characteristics are written to the standard output stream. .

Three transparent glasses are created in the line 4 to 10. A instance of the *GlassPanes* class is created in the line 13 with these transparent glasses. The heat transfer coefficients of air gap and the film coefficients at the glass surface are initialized in the line 16 to 21. In the line 24 to 28, the characteristics of the glass panes are written to the standard output stream.

The outside glass pane is replaced by heat reflecting glass in the line 32 for comparison of characteristics.

```
1  /// <summary>Sample program calculating the charateristics of a glass panes</summary>
2  private static void glassPanesTest()
3  {
4      //Create an array of the GlassPanes.Glass class
5      GlassPanes.Pane[] panes = new GlassPanes.Pane[3];
6
7      //Create a transparent glass (3mm).
8      panes[0] = new GlassPanes.Pane(0.79, 0.07, 0.85, 0.07, 131);    //Inside of the window
9      panes[1] = new GlassPanes.Pane(0.79, 0.07, 0.85, 0.07, 131);
10     panes[2] = new GlassPanes.Pane(0.79, 0.07, 0.85, 0.07, 131);    //Outside of the window
11
12     //Create an instance of GlassPanes class
13     GlassPanes glass = new GlassPanes(panes);
14
15     //Set heat transfer coefficients[W/(m2-K)] of air gap
16     glass.SetHeatTransferCoefficientsOfGaps(0, 1 / 0.12);
17     glass.SetHeatTransferCoefficientsOfGaps(1, 1 / 0.12);
18
19     //Set film coefficients[W/(m2-K)] at the surface of glass.
20     glass.SetInsideFilmCoefficient(9.26);     //Inside of the window
21     glass.SetOutsideFilmCoefficient(23.26);    //Outside of the window
22
23     //Check the characteristics of the glass panes.
24     Console.WriteLine("Transparent glass(3mm) * 3");
25     Console.WriteLine("Overall transmissivity[-] = " + glass.OverallTransmissivity.ToString("F2"));
26     Console.WriteLine("Overall absorptivity[-] = " + glass.OverallAbsorptivity.ToString("F2"));
27     Console.WriteLine("Heat transmission coefficient of glass[-] = " + glass.ThermalTransmittanceOfGlass.ToString("F2"));
28     Console.WriteLine("Heat transmission coefficient[-] = " + glass.ThermalTransmittance.ToString("F2"));
29     Console.WriteLine();
30
31     //Change the outside glass pane to heat reflecting glass(6mm)
32     panes[0] = new GlassPanes.Pane(GlassPanes.Pane.PredifinedGlassPane.HeatReflectingGlass06mm);
33
34     //Check the characteristics of a single glass pane.
35     Console.WriteLine("Heat reflecting glass(6mm)");
36     Console.WriteLine("Transmissivity[-] = " + panes[0].OuterSideTransmissivity.ToString("F2"));
37     Console.WriteLine("absorptivity[-] = " + panes[0].OuterSideAbsorptivity.ToString("F2"));
38     Console.WriteLine("Reflectivity[-] = " + panes[0].OuterSideReflectivity.ToString("F2"));
39     Console.WriteLine();
40
41     //Create an instance of GlassPanes class. Other properties are same as above
42     glass = new GlassPanes(panes);
43     glass.SetHeatTransferCoefficientsOfGaps(0, 1 / 0.12);
44     glass.SetHeatTransferCoefficientsOfGaps(1, 1 / 0.12);
45     glass.SetInsideFilmCoefficient(9.26);     //Inside of the window
46     glass.SetOutsideFilmCoefficient(23.26);    //Outside of the window
47
48     //Check the characteristics of the glass panes.
49     Console.WriteLine("Heat reflecting glass(6mm) + Transparent glass(3mm) * 2");
50     Console.WriteLine("Overall transmissivity[-] = " + glass.OverallTransmissivity.ToString("F2"));
51     Console.WriteLine("Overall absorptivity[-] = " + glass.OverallAbsorptivity.ToString("F2"));
52     Console.WriteLine("Heat transmission coefficient of glass[-] = " + glass.ThermalTransmittanceOfGlass.ToString("F2"));
53     Console.WriteLine("Heat transmission coefficient[-] = " + glass.ThermalTransmittance.ToString("F2"));
54
55     Console.Read();
56 }
```

Figure 6.4 The sample program which calculates the characteristics of a glass panes

Figure 6.5 shows the result of the simulation. The transmissivity of the second glass panes is smaller than that of the first glass panes, since the reflectivity of the heat reflecting glass is bigger than the normal transparent glass.

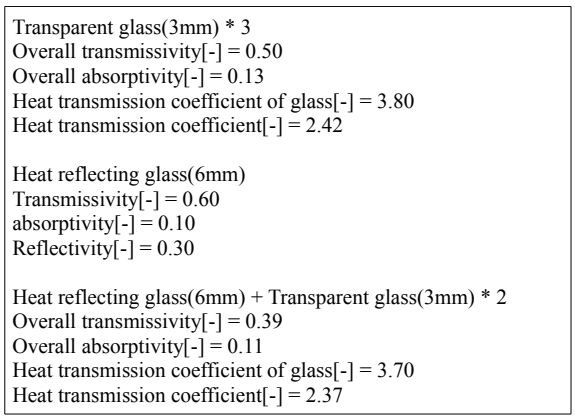


Figure 6.5 The result of the simulation

6.2.2 Window class

The *Window* class is a model of a window. It is initialized by *GlassPanes* object described in the section 6.2.1. Table 6.6 shows the constructors of the *Window* class.

Table6.6 The constructors of the Window class

No.	Argument 1	Argument 2	Argument 3
1	Multi-layered glass panes (ImmutableGlassPanes object)	-	-
2	Multi-layered glass panes (ImmutableGlassPanes object)	Inclined plane of outside surface (ImmutableIncline object)	-
3	Multi-layered glass panes (ImmutableGlassPanes object)	Inclined plane of outside surface (ImmutableIncline object)	Sun shade for the window (ImmutableSunShade object)

The second argument of the constructor is a Incline object described in the section 5.2.1. The third argument is a *SunShade* object which calculates a shading rate of the window. The *SunShade* class will be described later in the section 6.2.4.

Table 6.7 shows the properties defined in the *Window* class.

Table 6.7 The properties defined in the Window class.

Name	Description	Has set accessor	Unit	Type
AbsorbedHeatGain	Heat gain by absorption	-	W	double
Albedo	Albedo	TRUE	-	double
ConvectiveHeatGain	Radiative heat gain	-	W	double
Glass	Multi-layered glass	-	-	ImmutableGlassPanes
IndoorDrybulbTemperature	Indoor drybulb temperature	TRUE	°C	double
IndoorSurfaceTemperature	Indoor surface temperature	-	°C	double
NocturnalRadiation	Nocturnal radiation	TRUE	W/m^2	double
OutdoorDrybulbTemperature	Outdoor drybulg temperature	TRUE	°C	double
OutdoorSideIncline	Incline of outside surface	-	-	ImmutableIncline
OutdoorSurfaceTemperature	Outdoor surface temperature	-	°C	double
RadiativeHeatGain	Radiative heat gain	-	W	double
Shade	Sun shade of the window	TRUE	-	ImmutableSunShade
ShadowRate	Shadow rate	TRUE	-	double
Sun	Sun	TRUE	-	ImmutableSun
SurfaceArea	Surface are of the window	TRUE	m^2	double
TransferHeatGain	Heat gain by heat transfer	-	W	double
TransmissionHeatGain	Heat gain by transmission	-	W	double

Some boundary conditions (*Albedo*, *OutdoorDrybulbTemperature*, *IndoorDrybulbTemperature* and *NocturnalRadiation* properties) must be given to calculate a heat transfer through the window. An information about sun shade of the window is given as *Shade* property to calculate shadow rate, otherwise user should specify shadow rate directly by *ShadowRate* property. The information about solar radiations (direct normal radiation, diffuse horizontal radiation and global horizontal radiation) are given by *Sun* property. It is Sun object described at section 5.2.3.

When the boundary conditions are fixed, heat transfer (absorbed, transfer and transmission heat gain) through the window is determined. These heat gains are separated into two kinds of heat gains, convective heat gain and radiative heat gain. The convective heat gain is the heat gain that directly raise the temperature of a room. The radiative heat gain is the heat gain that first raise the temperature of room wall surfaces or other window surfaces. Therefore, the radiative heat gain has a time delay to the room temperature. Sum of the *AbsorbedHeatGain*, *TransferHeatGain* and *TransmissionHeatGain* is equal to *ConvectiveHeatGain* and

RadiativeHeatGain.

Figure 6.6 shows a sample program which calculates the heat gain from a window.

```
/// <summary>Sample program calculating the heat gain from the window</summary>
private static void windowTest()
{
    //A sample weather data
    //direct normal radiation [W/m2]
    double[] wdIdn = new double[] { 0, 0, 0, 0, 0, 244, 517, 679, 774, 829, 856, 862, 847, 809, 739, 619, 415, 97, 0, 0, 0, 0, 0, 0 };
    //diffuse horizontal radiation [W/m2]
    double[] wdIsky = new double[] { 0, 0, 0, 0, 0, 21, 85, 109, 116, 116, 113, 110, 109, 111, 114, 116, 114, 102, 63, 0, 0, 0, 0, 0, 0 };
    //drybulb temperature [C]
    double[] wdDbt = new double[] { 27, 27, 27, 27, 27, 28, 29, 30, 31, 32, 32, 33, 33, 33, 34, 33, 32, 32, 31, 30, 29, 29, 28, 28 };
    //nocturnal radiation [W/m2]
    double[] wdRN = new double[] { 24, 24, 24, 24, 24, 24, 25, 25, 25, 25, 26, 26, 26, 26, 26, 26, 26, 25, 25, 25, 25, 24, 24, 24 };

    //Create a window with a single 3mm transparent glass pane
    GlassPanes.Pane pane = new GlassPanes.Pane(GlassPanes.Pane.PredifinedGlassPane.TransparentGlass03mm);
    GlassPanes glassPane = new GlassPanes(pane);
    Window window = new Window(glassPane);

    //Set wall surface information
    WindowSurface outsideWindowSurface = window.GetSurface(true);
    outsideWindowSurface.FilmCoefficient = 23d;
    outsideWindowSurface.Albedo = 0.2;
    WindowSurface insideWindowSurface = window.GetSurface(false);
    insideWindowSurface. FilmCoefficient = 9.3;

    //Set incline of an outdoor surface : South, vertical incline
    window.OutSideIncline = new Incline(Incline.Orientation.S, 0.5 * Math.PI);

    //There is no sun shade
    window.Shade = SunShade.EmptySunShade;

    //Initialize sun. Tokyo : 7/21 0:00
    Sun sun = new Sun(Sun.City.Tokyo);
    DateTime dTime = new DateTime(2001, 7, 21, 0, 0, 0);
    sun.Update(dTime);
    window.Sun = sun;

    //Indoor drybulb temperature is constant (25C)
    window.IndoorDrybulbTemperature = 25;

    //Result : Title line
    Console.WriteLine(" Time |Transmission[W]|Absorption[W]|Transfer[W]|Convective[W]|Radiative[W]");

    //execute simulation
    for (int i = 0; i < 24; i++)
    {
        //Set radiations (calculate global horizontal radiation from direct normal and diffuse horizontal radiation)
        sun.SetGlobalHorizontalRadiation(wdIsky[i], wdIdn[i]);
        //Set nocturnal radiation
        window.NocturnalRadiation = wdRN[i];
        //Set outdoor temperature
        window.OutdoorDrybulbTemperature = wdDbt[i];

        //Output result
        Console.WriteLine(dTime.ToShortTimeString().PadLeft(5) + " | " + window.TransmissionHeatGain.ToString("F1").PadLeft(13) + " | " +
        window.AbsorbedHeatGain.ToString("F1").PadLeft(11) + " | " + window.TransferHeatGain.ToString("F1").PadLeft(9) + " | " +
        window.ConvectiveHeatGain.ToString("F1").PadLeft(11) + " | " + window.RadiativeHeatGain.ToString("F1").PadLeft(11));

        //Update time
        dTime = dTime.AddHours(1);
        sun.Update(dTime);
    }

    Console.Read();
}
```

Figure 6.6 The sample program which calculates the heat gain from the window

The arrays declared in the line 4~12 is a sample weather data. These are the value of direct normal radiation, diffuse horizontal radiation, drybulb temperature and nocturnal radiation of one day.

An instance of the *GlassPanes* class is created in the line 15 and 16. The transparent 3mm glass pane is created. A *Window* object is initialized with this *GlassPanes* object in the line 17.

A *WindowSurface* object is used to set surface state in the line 20~24. Overall heat transfer coefficients and albedo are set to *WindowSurface* object. Albedo is solar radiation reflectivity of ground surface. In urban areas, value of albedo is around 0.2.

Outdoor surface incline is set in the line 27. It is the south vertical window.

The sun shade is set in the line 30. In this sample, the window has no sun shade. The *SunShade* class will be described late in the section 6.2.4.

A instance of *Sun* class is created in the line 33~36. The location is Tokyo and the date is 7/21. It is 0:00 at the beginning of the simulation. The line from 45 to 62 is the iteration. It iterates 24 times to simulate 24 hours.

The three boundary conditions, solar radiation, nocturnal radiation and drybulb temperature is set in the line 47~52. The heat gains from the window is calculated since boundary conditions are fixed. All the heat gains are written to the standard output stream in the line 55~57. Finally, in the line 60 and 61, 1 hour is added to update the position of the sun.

Figure 6.7 shows the result of the simulation.

Time	Transmission[W]	Absorption[W]	Transfer[W]	Convective[W]	Radiative[W]
0:00	0.0	0.0	12.9	5.8	7.1
1:00	0.0	0.0	12.9	5.8	7.1
2:00	0.0	0.0	12.9	5.8	7.1
3:00	0.0	0.0	12.9	5.8	7.1
4:00	9.7	0.3	13.2	6.0	7.4
5:00	40.3	1.1	20.4	9.7	11.8
6:00	60.4	1.6	27.4	13.0	15.9
7:00	77.0	2.0	34.3	16.3	20.0
8:00	91.7	2.4	41.2	19.6	24.0
9:00	134.4	3.5	48.7	23.5	28.7
10:00	190.5	5.0	50.2	24.8	30.4
11:00	229.5	6.0	57.7	28.7	35.0
12:00	236.3	6.2	57.9	28.8	35.2
13:00	208.2	5.4	57.1	28.2	34.4
14:00	154.5	4.0	62.2	29.8	36.4
15:00	97.9	2.6	54.2	25.6	31.2
16:00	64.8	1.7	46.9	21.9	26.7
17:00	31.9	0.8	46.1	21.1	25.8
18:00	0.0	0.0	38.8	17.4	21.3
19:00	0.0	0.0	32.3	14.5	17.8
20:00	0.0	0.0	25.8	11.6	14.2
21:00	0.0	0.0	25.8	11.6	14.2
22:00	0.0	0.0	19.4	8.7	10.7
23:00	0.0	0.0	19.4	8.7	10.7

Figure 6.7 The result of the simulation

6.2.3 AirFlowWindow class

****

6.2.4 SunShade class

The *SunShade* class is a model of a sun shade. Figure 6.8 shows the shapes of the sun shade that can be treated by *SunShade* class. To create these shaped sun shades, some static methods are defined in the *SunShade* class. These static methods must be used to create an instance of the *SunShade* class, since *SunShade* class doesn't have constructor. There is special static member, *EmptySunShade* property, to express empty sun shade.

Table 6.8 shows the properties defined in the *SunShade* class. Most of the properties are read only property.

Incline is described in the section 5.2.1. *SunShade* class can calculate a shadow rate at inclined plane like roof skylight.

An opening area can be reversed with *IsReverse* property. If the *IsReverse* property is true, the opening area is treated as the sun shade and the sun shade is treated as opening area. For example, we can calculate shadow rate of a dry area by reversing the horizontal sun shade (Figure 6.9).

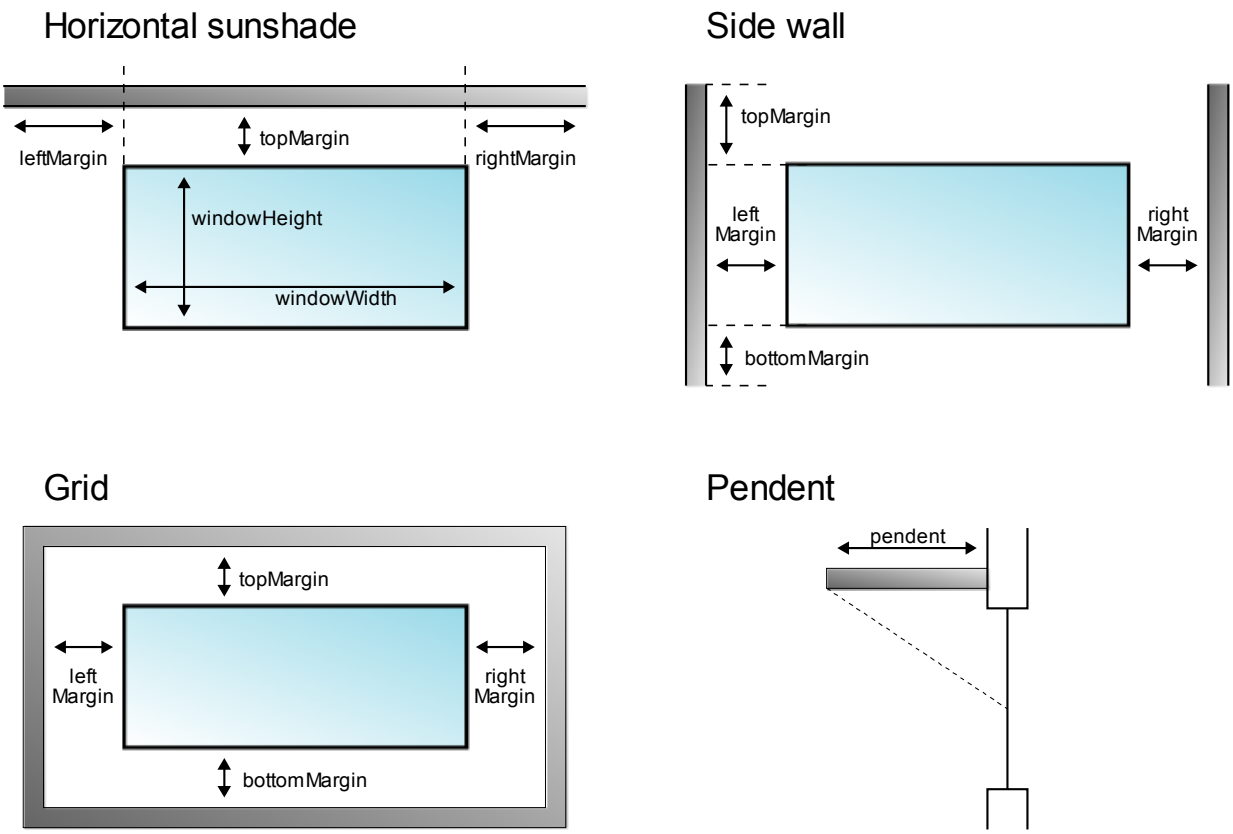


Figure 6.8 Three types of the sun shade

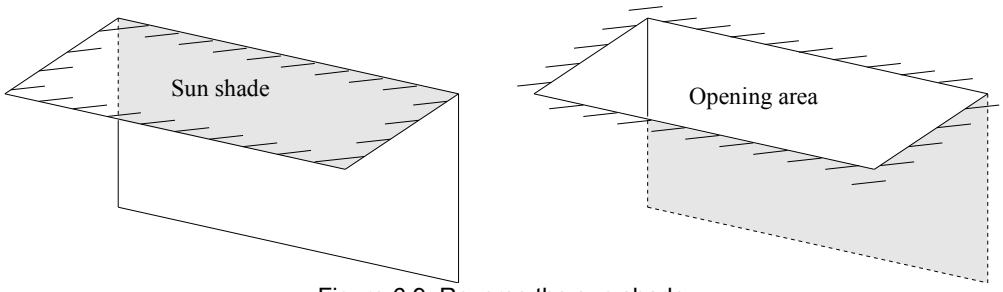


Figure 6.9 Reverse the sun shade

Table 6.8 The properties defined in the SunShade class

Name	Description	Has set accessor	Unit	Type
BottomMargin	Bottom margin	-	m	double
Incline	Incline	TRUE	-	ImmutableIncline
IsReverse	Whether sun shade is reversed or not	TRUE	-	bool
LeftMargin	Length of left margin	-	m	double
Name	Name of sun shade	TRUE	-	string
OverHang	Length of overhang	-	m	double
RightMargin	Length of right margin	-	m	double
SunShadeShape	Shape type	-	-	SunShade.Shape
TopMargin	Length of top margin	-	m	double
WindowHeight	Height of the window	-	m	double
WindowWidth	Width of the window	-	m	double

Table 6.9 The static methods defined in the SunShade class

Name	Description			
MakeGridSunShade	Function	Create grid sun shade.		
	Return	grid sun shade.		
	Arg.1	Window width [m]	Arg.2	Window height [m]
	Arg.3	Length of overhang [m]	Arg.4	Length of left margin [m]
	Arg.5	Length of right margin [m]	Arg.6	Length of top margin [m]
	Arg.7	Length of bottom margin [m]	Arg.8	Incline
MakeHorizontalSunShade	Function	Create horizontal sun shade.		
	Return	horizontal sun shade.		
	Arg.1	Window width [m]	Arg.2	Window height [m]
	Arg.3	Length of overhang [m]	Arg.4	Length of left margin [m]
	Arg.5	Length of right margin [m]	Arg.6	Length of top margin [m]
	Arg.7	Incline	-	-
MakeHorizontalSunShade	Function	Create horizontal (infinite length) sun shade.		
	Return	horizontal (infinite length) sun shade.		
	Arg.1	Window width [m]	Arg.2	Window height [m]
	Arg.3	Length of overhang [m]	Arg.4	Length of top margin [m]
	Arg.5	Incline	-	-
MakeVerticalSunShade	Function	Create left or right side wall sun shade		
	Return	left or right side wall sun shade		
	Arg.1	Window width [m]	Arg.2	Window height [m]
	Arg.3	Length of overhang [m]	Arg.4	Length of side [m]
	Arg.5	Set true if left side wall	Arg.6	Length of top margin[m]
	Arg.7	Length of bottom margin [m]	Arg.8	Incline
MakeVerticalSunShade	Function	Create left or right side wall (infinite length) sun shade		
	Return	eft or right side wall (infinite length) sun shade		
	Arg.1	Window width [m]	Arg.2	Window height [m]
	Arg.3	Length of overhang [m]	Arg.4	Length of side margin [m]
	Arg.5	Set true if left side wall	Arg.6	Incline
MakeVerticalSunShade	Function	Create both side wall sun shade		
	Return	both side wall sun shade		
	Arg.1	Window width [m]	Arg.2	Window height [m]
	Arg.3	Length of overhang [m]	Arg.4	Length of side margin [m]
	Arg.5	Length of top margin [m]	Arg.6	Length of bottom margin [m]
	Arg.7	Incline	-	-
MakeVerticalSunShade	Function	Create both side wall (infinite length) sun shade		
	Return	both side wall (infinite length) sun shade		
	Arg.1	Window width [m]	Arg.2	Window height [m]
	Arg.3	Length of overhang [m]	Arg.4	Length of side margin [m]
	Arg.5	Incline	-	-

6.2.5 WallLayers class

The most of the building walls consist of more than one materials (structure, insulation and covering material). The *WallLayers* class is a model of a multi layered wall. Table 6.10 and 6.11 shows the properties and methods defined in the *WallLayers* class.

Each layer can be added or removed by *AddLayer* method and *RemoveLayer* method. After the composition of the wall layer is fixed, thermal transmission of the wall layer can be calculated by *GetThermalTransmission* method.

WallLayers.*Layer* class is defined in the *WallLayers* class to make a single wall layer. Figure 6.12 shows the constructors of the *WallLayers*.*Layer* class. The wall layer should be split when the layer is very thick.

Table 6.10 The properties defined in the WallLayers class

Name	Description	Has set accessors	Unit	Type
LayerNumber	Number of layers	-	-	uint
Name	Name	TRUE	-	string

Table 6.11 The methods defined in the WallLayers

Name	Description			
GetThermalTransmission	Function	Get thermal transmission [W/(m^{2}K)] of wall layers		
	Return	ThermalTransmission[W/(m^{2}K)]		
	Arg.1	Film coefficient at surface 1 [W/(m^{2}K)]	Arg.2	Film coefficient at surface 2 [W/(m^{2}K)]
GetThermalTransmission	Function	Get thermal transmission [W/(m^{2}K)] of wall layers		
	Return	ThermalTransmission[W/(m^{2}K)]		
	Arg.1	-	Arg.2	-
UsingMaterial	Function	Return whether wall material is used in this wall layer or not		
	Return	Whether wall material is used in this wall layer or not		
	Arg.1	Material (ImmutableWallMaterial type)	-	-
AddLayer	Function	Add a wall layer to the wall layers		
	Return	-		
	Arg.1	The wall layer (Layer type)	Arg.2	-
GetLayer	Function	Get the lists of the wall layers		
	Return	Array list of the wall layers (Layer array)		
	Arg.1	-	Arg.2	-
GetLayer	Function	Get the wall layer		
	Return	The wall layer (Layer type)		
	Arg.1	Index number of the wall layer	Arg.2	-
RemoveLayer	Function	Remove the layer from the wall layers		
	Return	-		
	Arg.1	Index number of the wall layer	Arg.2	-
ReplaceLayer	Function	Replace a wall layer		
	Return	-		
	Arg.1	Index number of the wall layer	Arg.2	The wall layer (Layer type)

Table 6.12 The constructors of the WallLayers.Layer class

No.	Argument 1	Argument 2	Argument 3
1	Material (ImmutableWallMaterial type)	Thickness of the layer (double type)	-
2	Material (ImmutableWallMaterial type)	Thickness of the layer (double type)	split numbers (uint type)

The *WallMaterial* class is a model of wall material.

Table 6.13 shows the constructors of the *WallMaterial* class. The principal information about wall material is name, thermal conductivity [W/(m K)] and volumetric specific heat [kJ/(m^3 K)]. These are the arguments of the first constructor. When a material has no heat capacity (like air), set 0 to the first argument and set a heat conductance [W/(m^2 K)] to the third argument.

An instance of the *WallMaterial* class can be also initialized with predefined commonly used materials (the second constructor). Table 6.15 shows the materials predefined in the *WallMaterial* class.

Table 6.14 shows the properties of the *WallMaterial* class.

Table 6.13 The constructors of the WallMaterial class

No.	Argument 1	Argument 2	Argument 3
1	Name of the material (string type)	thermal conductivity [W/(m K)] (double type)	volumetric specific heat [kJ/(m^3 K)] (double type)
2	Predefined wall material (WallMaterial.PredifinedMaterials enumerator)	-	-
3	Wall material (ImmutableWallMaterial type)	-	-

Table 6.14 The properties of the WallMaterial class

Name	Description	Has set accessor	Unit	Type
Material	predefined material	-	-	WallMaterial.PredifinedMaterials
Name	Name	TRUE	-	string
ThermalConductivity	thermal conductivity	TRUE	W/(m K)	double
VolumetricSpecificHeat	volumetric specific heat	TRUE	kJ/(m^3 K)	double

The thermal transmission [W/(m^2 K)] of a wall can be calculated from thermal conductivity [W/(m K)] and thickness [m] of each wall layer according to equation 6.10.

$$K = 1/\left\{\frac{1}{\alpha_1} + \sum \frac{l}{\lambda} + \sum \frac{1}{C_{air}} + \frac{1}{\alpha_2}\right\} \tag{6.10}$$

K : thermal transmittance of a wall [W/(m^2 K)], α_1 : film coefficient at wall surface 1 [W/(m^2 K)]
α_2 : film coefficient at wall surface 2 [W/(m^2 K)], λ : thermal conductivity [W/(m K)], l : thickness [m]
C_{air} : heat conductance of air layer [W/(m^2 K)]

Table 6.15 Members of the PredifinedMaterials enumerator

Name	thermal conductivity	volumetric specific heat	Name	thermal conductivity	volumetric specific heat
Mortar	1.512	1591	SprayedRockWool	0.047	167.9
ReinforcedConcrete	1.6	1896	BeadMethodPolystyreneFoam_S	0.034	33.9
LightweightAggregateConcrete1	0.81	1900	BeadMethodPolystyreneFoam_1	0.036	37.7
LightweightAggregateConcrete2	0.58	1599	BeadMethodPolystyreneFoam_2	0.037	31.4
AutomaticLevelControl	0.17	661.4	BeadMethodPolystyreneFoam_3	0.04	25.1
Brick	0.62	1386	BeadMethodPolystyreneFoam_4	0.043	18.8
FireBrick	0.99	1553	ExtrudedPolystyreneFoam_1	0.04	25.1
Copper	370.1	3144	ExtrudedPolystyreneFoam_2	0.034	25.1
Aluminum	200	2428	ExtrudedPolystyreneFoam_3	0.028	25.1
Steel	53.01	3759	RigidUrethaneFoam_1_1	0.024	56.1
Lead	35.01	1469	RigidUrethaneFoam_1_2	0.024	44
StainlessSteel	15	3479	RigidUrethaneFoam_1_3	0.026	31.4
FloatGlass	1	1914	RigidUrethaneFoam_2_1	0.023	56.1
PolyvinylChloride	0.17	1023	RigidUrethaneFoam_2_2	0.023	44
Wood1	0.12	519.1	RigidUrethaneFoam_2_3	0.024	31.4
Wood2	0.15	648.8	RigidUrethaneFoam_InSite	0.026	49.8
Wood3	0.19	845.6	PolyethyleneFoam_A	0.038	62.8
Plywood	0.19	716	PolyethyleneFoam_B	0.042	62.8
WoodWoolCement	0.1	841.4	PhenolicFoam_1_1	0.033	37.7
WoodChipCement	0.17	1679	PhenolicFoam_1_2	0.03	37.7
HardBoard	0.17	1233	PhenolicFoam_2_1	0.036	56.5
ParticleBoard	0.15	715.8	PhenolicFoam_2_2	0.034	56.5
PlasterBoard	0.17	1030	InsulationBoard_A	0.049	324.8
GypsumPlaster	0.6	1637	TatamiBoard	0.045	15.1
WhiteWash	0.7	1093	SheathingInsulationBoard	0.052	390.1
SoilWall	0.69	1126	CelluloseFiberInsulation_1	0.04	37.7
FiberCoating	0.12	4.2	CelluloseFiberInsulation_2	0.04	62.8
Tatami	0.11	527.4	Soil	1.047	3340
Tile	1.3	2018	ExpandedPolystyrene	0.035	300
PlasticTile	0.19	4.2	CoveringMaterial	0.14	1680
GlassWoolInsulation_10K	0.05	8.4	Linoleum	0.19	1470
GlassWoolInsulation_16K	0.045	13.4	Carpet	0.08	318
GlassWoolInsulation_24K	0.038	20.1	AsbestosPlate	1.2	1820
GlassWoolInsulation_34K	0.036	26.8	SealedAirGap	5.8	0
HighGradeGlassWoolInsulation_16K	0.038	13.4	AirGap	11.6	0
HighGradeGlassWoolInsulation_24K	0.036	20.1	PolystyreneFoam	0.035	80
BlowingGlassWoolInsulation_13K	0.052	10.9	StyreneFoam	0.035	10
BlowingGlassWoolInsulation_18K	0.052	16.7	RubberTile	0.4	784
BlowingGlassWoolInsulation_30K	0.04	29.3	Kawara	1	1506
BlowingGlassWoolInsulation_35K	0.04	37.7	LightweightConcrete	0.78	1607
RockWoolInsulationMat	0.038	33.5	Asphalt	0.11	920
RockWoolInsulationFelt	0.038	41.9	FrexibleBoard	0.35	1600
RockWoolInsulationBoard	0.036	58.6	CalciumSilicateBoard	0.13	680
BlowingRockWoolInsulation_25K	0.047	20.9	PhenolicFoam	0.02	37.7
BlowingRockWoolInsulation_35K	0.051	29.3	GNT	4.3	2.9
RockWoolAcousticBoard	0.058	293.9	AcrylicResin	0.21	1666

Figure 6.9 shows a sample program which calculates the thermal transmission of a multi-layered wall.

The wall layers consists of plywood, air gap, reinforced concrete and white wash. These materials are created in the line 7~16. The first three materials are created with using *PredefinedMaterials* enumerator. The last material (white wash) is created by giving thermal conductivity and volumetric specific heat directly to the constructor (line 16).

A instance of the WallLayers.Layer class is created in the line 18~26. The thickness of each layer is 20mm, 10mm, 150mm and 10mm. Although heat conductance of the air gap doesn't depends on the thickness.

The name and thickness of each layer and the thermal transmission of the wall layers are written to standard output stream in the line 28~36.

The reinforced concrete layer is replaced by lightweight concrete in the line 39 for comparison.

Figure 6.11 shows the result of the simulation.

```
/// <summary>Sample program calculating the thermal transimission of the wall layers</summary>
private static void wallLayersTest()
{
    //Create an instance of WallLayers
    WallLayers wLayers = new WallLayers("Sample wall layer");

    //Make an array of materials
    WallMaterial[] materials = new WallMaterial[4];
    //The first layer : plywood
    materials[0] = new WallMaterial(WallMaterial.PredefinedMaterials.Plywood);
    //The second layer : air gap
    materials[1] = new WallMaterial(WallMaterial.PredefinedMaterials.AirGap);
    //The thirg layer : concrete
    materials[2] = new WallMaterial(WallMaterial.PredefinedMaterials.ReinforcedConcrete);
    //The fourth layer : white Wash
    materials[3] = new WallMaterial("White Wash", 0.7, 1000);

    //Add a layer to WallLayers object
    //plywood : 20mm
    wLayers.AddLayer(new WallLayers.Layer(materials[0], 0.02));
    //air gap : heat conductance doesn't depend on thickness
    wLayers.AddLayer(new WallLayers.Layer(materials[1], 0.01));
    //concrete : 150mm
    wLayers.AddLayer(new WallLayers.Layer(materials[2], 0.15));
    //white Wash : 10mm
    wLayers.AddLayer(new WallLayers.Layer(materials[3], 0.01));

    //output result
    Console.WriteLine("Wall composition");
    for (uint i = 0; i < wLayers.LayerNumber; i++)
    {
        WallLayers.Layer layer = wLayers.GetLayer(i);
        Console.WriteLine("Layer " + (i + 1) + " : " + layer.Material.Name + "(" + layer.Thickness + "m)");
    }
    Console.WriteLine("Thermal transmission = " + wLayers.GetThermalTransmission().ToString("F1") + " W/(m2-K)");
    Console.WriteLine();

    //Replace concrete to light weight concrete
    wLayers.ReplaceLayer(2, new WallLayers.Layer(new WallMaterial(WallMaterial.PredefinedMaterials.LightweightConcrete), 0.15));

    //output result
    Console.WriteLine("Wall composition");
    for (uint i = 0; i < wLayers.LayerNumber; i++)
    {
        WallLayers.Layer layer = wLayers.GetLayer(i);
        Console.WriteLine("Layer " + (i + 1) + " : " + layer.Material.Name + "(" + layer.Thickness + "m)");
    }
    Console.WriteLine("Thermal transmission = " + wLayers.GetThermalTransmission().ToString("F1") + " W/(m2-K)");
    Console.WriteLine();

    Console.Read();
}
```

Figure 6.10 The sample program which calculates thermal transmission of the wall layers

```
Wall composition
Layer 1 : Plywood(0.02m)
Layer 2 : Air Gap(999m)
Layer 3 : Reinforced Concrete(0.15m)
Layer 4 : White Wash(0.01m)
Thermal transmission = 3.3 W/(m2-K)

Wall composition
Layer 1 : Plywood(0.02m)
Layer 2 : Air Gap(999m)
Layer 3 : Lightweight Concrete(0.15m)
Layer 4 : White Wash(0.01m)
Thermal transmission = 2.5 W/(m2-K)
```

Figure 6.11 The result of the simulation

6.2.6 Wall class

The *Wall* class is a model of a wall. It can calculate unsteady state heat conduction. Table 6.16 shows the constructors of the *Wall* class. A *Wall* object is initialized with a *WallLayer* object described in the section 6.2.5.

Table 6.17 and 6.18 shows the properties and the methods defined in the *Wall* class.

Table 6.16 The constructors of the Wall class

No.	Argument 1	Argument 2
1	Wall compostion (ImmutableWallLayers type)	Name of the wall (string type)
2	Wall compostion (ImmutableWallLayers type)	-

table 6.17 The properties defined in the Wall class

Name	Description	Has set accessor	Unit	Type
TimeStep	Calculating time step	TRUE	sec	double
SurfaceArea	Surface area of wall	TRUE	m^2	double
Layers	WallLayers object	-	-	-
AirTemperature1	Air temperature at wall surface 1	TRUE	°C	double
AirTemperature2	Air temperature at wall surface 2	TRUE	°C	double
Radiation1	Radiation at wall surface 1	TRUE	W/m^2	double
Radiation2	Radiation at wall surface 2	TRUE	W/m^2	double
Incline1	Incline at wall surface 1	-	-	-
Incline2	Incline at wall surface 2	-	-	-

1) Unsteady heat conduction of normal wall

Two principal boundary conditions are air temperature and radiation at wall surface. These conditions can be set by *AirTemperature* and *Radiation* property of the *Wall* class. If the boundary conditions are fixed, the state of the wall can be calculated with *Update* method. Once the *Update* methods is called, time given by *TimeStep* property passes.

The temperature of each wall layer is determined by solving equation 6.11 [2]. As shown in figure 6.12, heat capacities are concentrated into the boundary of wall layers.

Figure 6.13 shows a sample program which calculates unsteady heat conduction with *Wall* class. Table 6.19 shows the compositions of the wall. It is as same as wall described in the section 3.3.2, which was solved by the circuit networks.

Figure 6.14 and 6.15 shows the results of the simulation. It seems that the results is as same as that of the simulation solved by the circuit networks.

$$\left.\frac{\partial T}{\partial t}\right|_m = \frac{1}{0.5(CAP_m + CAP_{m+1})}\left\{\frac{1}{R_{m+1}}(T_{m+1}-T_m)-\frac{1}{R_m}(T_m-T_{m-1})\right\} \tag{6.11}$$

T : temperature of wall [°C], t : time [sec], M : number of the layer,
CAP : heat capacity per surface area [J/(m²K)], R : thermal resistance [(m²K)/W]

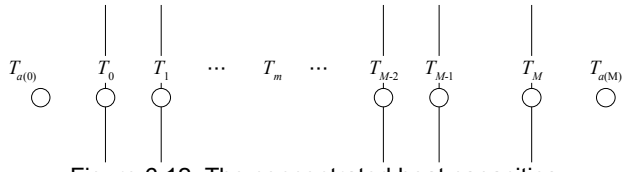


Figure 6.12 The concentrated heat capacities

Table 6.18 The methods defined in the Wall class

Name	Description			
Update	Function	Update the state of the wall		
	Return	-		
	Arg.1	-	Arg.2	-
SetIncline	Function	Set incline to wall surface. Incline of opsite side surface is automatically set.		
	Return	-		
	Arg.1	Incline (ImmutableIncline type)	Arg.2	Is target wall suraface 1 or not (bool type)
GetSurface	Function	Get wall suraface object		
	Return	Wall suraface object		
	Arg.1	Is target wall suraface 1 or not (bool type)	Arg.2	-
InitializeTemperature	Function	Initialize inside temperature of the wall		
	Return	-		
	Arg.1	Temperature of wall (double type)	Arg.2	-
SetFilmCoefficient	Function	Set film coefficient of wall surface [W/(m^2 K)]		
	Return	-		
	Arg.1	film coefficient (double type)	Arg.2	Is target wall suraface 1 or not (bool type)
SetFilmCoefficient	Function	Set film coefficient of wall surface [W/(m^2 K)]		
	Return	-		
	Arg.1	film coefficient of surface 1 (double type)	Arg.2	film coefficient of surface 2 (double type)
GetFilmCoefficient	Function	Get film coefficient of wall surface [W/(m^2 K)]		
	Return	Film coefficient of wall surface		
	Arg.1	Is target wall suraface 1 or not (bool type)	Arg.2	-
SetConvectiveRate	Function	Set convective rate of film coefficient [-]		
	Return	-		
	Arg.1	convective rate of film coefficient [-]	Arg.2	-
SetRadiativeRate	Function	Set radiative rate of film coefficient [-]		
	Return	-		
	Arg.1	radiative rate of film coefficient [-]	Arg.2	-
GetConvectiveRate	Function	Get convective rate of film coefficient [-]		
	Return	convective rate of film coefficient [-]		
	Arg.1	-	Arg.2	-
GetRadiativeRate	Function	Get radiative rate of film coefficient [-]		
	Return	radiative rate of film coefficient [-]		
	Arg.1	-	Arg.2	-

Table 6.19 Wall composition

Number	Material	Thermal conductivity [W/(m K)]	Volumetric specific heat [kJ/(m^3 K)]	Thickness [mm]
1	Plywood	0.19	716	25
2	Concrete	1.4	1934	120
3	Air gap	Thermal resistance = 0.086 (m^2 K)/W		
4	Rock wool	0.042	84	50

```
/// <summary>Sample program calculating the unsteady heat conduction of wall</summary>
private static void wallTest1()
{
    WallLayers layers = new WallLayers();
    WallLayers.Layer layer;
    layer = new WallLayers.Layer(new WallMaterial("Plywood", 0.19, 716), 0.025);
    layers.AddLayer(layer);
    layer = new WallLayers.Layer(new WallMaterial("Concrete", 1.4, 1934), 0.120);
    layers.AddLayer(layer);
    layer = new WallLayers.Layer(new WallMaterial("Air gap", 1d / 0.086, 0), 0.020);
    layers.AddLayer(layer);
    layer = new WallLayers.Layer(new WallMaterial("Rock wool", 0.042, 84), 0.050);
    layers.AddLayer(layer);
    Wall wall = new Wall(layers);

    wall.TimeStep = 3600;
    wall.AirTemperature1 = 20;
    wall.AirTemperature2 = 10;
    wall.InitializeTemperature(10); //Initial temperature is 10 C
    wall.SurfaceArea = 1;

    Console.WriteLine("Plywood, Concrete, Air gap, Rock wool");
    double[] temps;
    for (int i = 0; i < 24; i++)
    {
        wall.Update();
        temps = wall.GetTemperatures();
        Console.Write((i + 1).ToString("F0").PadLeft(2) + "Hour | ");
        for (int j = 0; j < temps.Length - 1; j++) Console.Write(((temps[j] + temps[j + 1]) / 2d).ToString("F1") + " | ");
        Console.WriteLine();
    }

    //Iterate until wall become steady state
    for (int i = 0; i < 1000; i++) wall.Update();
    Console.WriteLine();
    Console.WriteLine("Steady state");
    temps = wall.GetTemperatures();
    for (int j = 0; j < temps.Length - 1; j++) Console.Write(((temps[j] + temps[j + 1]) / 2d).ToString("F1") + " | ");

    Console.WriteLine();
    Console.WriteLine("Heat transfer at steady state 1: " + wall.GetHeatTransfer(true).ToString("F1"));
    Console.WriteLine("Heat transfer at steady state 2: " + wall.GetHeatTransfer(false).ToString("F1"));
    Console.WriteLine("Heat transfer at steady state 3: " + wall.GetStaticHeatTransfer().ToString("F1"));

    Console.Read();
}
```

Figure 6.13 The sample program which calculates unsteady heat conduction of normal wall

```
Plywood, Concrete, Air gap, Rock wool
 1Hour | 12.9 | 10.5 | 10.2 | 10.1 |
 2Hour | 13.7 | 11.0 | 10.5 | 10.3 |
 3Hour | 14.2 | 11.5 | 10.9 | 10.4 |
 4Hour | 14.6 | 11.9 | 11.2 | 10.6 |
 5Hour | 14.9 | 12.3 | 11.6 | 10.9 |
 6Hour | 15.1 | 12.7 | 12.0 | 11.1 |
 7Hour | 15.4 | 13.1 | 12.4 | 11.2 |
 8Hour | 15.6 | 13.4 | 12.7 | 11.4 |
 9Hour | 15.9 | 13.7 | 13.1 | 11.6 |
10Hour | 16.1 | 14.0 | 13.4 | 11.8 |
11Hour | 16.2 | 14.3 | 13.6 | 11.9 |
12Hour | 16.4 | 14.6 | 13.9 | 12.0 |
13Hour | 16.6 | 14.8 | 14.2 | 12.2 |
14Hour | 16.7 | 15.0 | 14.4 | 12.3 |
15Hour | 16.9 | 15.2 | 14.6 | 12.4 |
16Hour | 17.0 | 15.4 | 14.8 | 12.5 |
17Hour | 17.2 | 15.6 | 15.0 | 12.6 |
18Hour | 17.3 | 15.8 | 15.2 | 12.7 |
19Hour | 17.4 | 16.0 | 15.4 | 12.8 |
20Hour | 17.5 | 16.1 | 15.5 | 12.9 |
21Hour | 17.6 | 16.3 | 15.7 | 13.0 |
22Hour | 17.7 | 16.4 | 15.8 | 13.1 |
23Hour | 17.8 | 16.5 | 16.0 | 13.1 |
24Hour | 17.8 | 16.6 | 16.1 | 13.2 |

Steady state
19.0 | 18.3 | 17.8 | 14.1 |
Heat transfer at steady state 1: 5.9
Heat transfer at steady state 2: -5.9
Heat transfer at steady state 3: 5.9
```

Figure 6.14 The result of the simulation

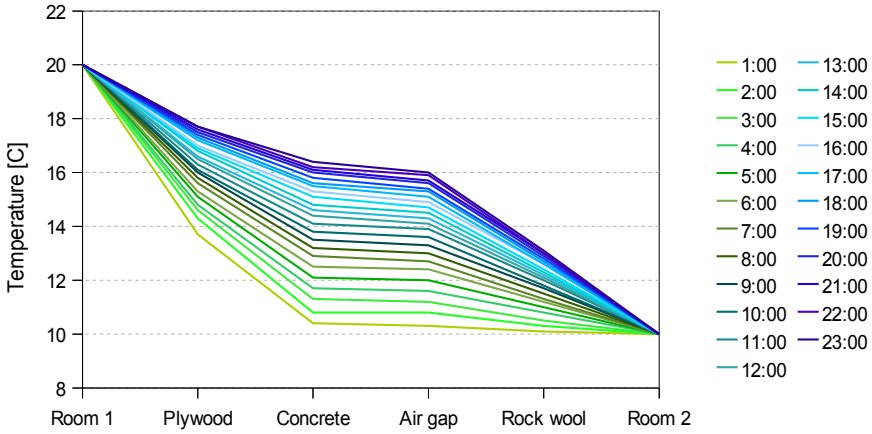


Figure 6.15 Distribution of wall temperatures

2) Unsteady heat conduction of wall with heating and cooling tube

Cooling tubes and heating tubes are installed in the floor or ceiling in radiant heating and cooling systems. The *Tube* class is a model to calculate this phenomenon.

Below is the constructor of the Tube class.

Tube tube = new Tube(epsilon, finEfficiency, fluidSpecificHeat);

The first argument (epsilon) is heat exchanger effectiveness[-] between temperature of fluid and temperature of wall layer where tube is installed. The second argument is fin efficiency[-]. The third argument is specific heat[J/(kg K)] of fluid. These value are obtained from equation 6.12~6.17.

$$\frac{H_P}{A_F} = \frac{\varepsilon_{PNL}(c_w G_w)}{A_F}(T_{wi} - T_m) \tag{6.12}$$

$$\varepsilon_{PNL} = \frac{\varepsilon_{PX}}{1 + (\varepsilon_{PX} c_w G_w / AC_f)(1/\eta_P - 1)} \tag{6.13}$$

$$\varepsilon_{PX} = 1 - \exp\left(-\frac{K_P A_F}{c_w G_w}\right) \tag{6.14}$$

$$\eta_P = \frac{1}{w}\left\{ D + (w - D)\frac{\tanh Z}{Z} \right\}$$
$$Z = \frac{w - D}{2}\sqrt{\frac{C_f}{\lambda D}} \tag{6.15}$$

$$K_p = 1/\left\{ A_F / L\left(1/\pi D \alpha_w + R_b\right)\right\} \tag{6.16}$$

$$a_w = 1057(1.352 + 0.0198 T_m) v^{0.8} / D^{0.2} \tag{6.17}$$

H_P : heat generation from floor [W], AF : surface area of floor [m^3], c_w : specific heat of fluid [J/kg], G_w : fluid flow rate [kg/s]
T_{wi} : inlet temperature of the fluid [°C], T_m : average temperature of wall layer where the tube is installed [°C]
ε_{PNL} : heat exchanger effectiveness between T_m and T_{wi} [-], ε_{PX} : heat exchanger effectiveness between surface temperature and T_{wi} [-]
Cf : thermal conductance from wall layer where the tube is installed and to the next layer[W/(m^2 K)]
η_P : fin efficiency [-], K_P : thermal transmission from surface temperature to fluid [W/(m^2 K)], w : installing span of tube [m]
D : diameter of the tube [m], λ : heat conductance of the floor [W/(m K)], L : length of the tube [m]
α_w : convective heat transfer coefficient at inner tube surface [W/(m^2 K)], R_b : thermal resistance between tube and floor panel [m K/W]
T_m : average temperature of the fluid [°C], v : fluid flow speed [m/s]

Table 6.20 The methods defined in the Tube class

Name	Description			
SetFlowRate	Function	Set flow rate [kg/s] of the fluid		
	Return	-		
	Arg.1	flow rate [kg/s] of the fluid	Arg.2	-
GetOutletFluidTemperature	Function	Calculate outlet fluid temperature [°C] from the heat transfer [W] to the tube.		
	Return	Outlet fluid temperature [°C]		
	Arg.1	heat transfer [W] to the tube	Arg.2	-
GetOutletFluidTemperature	Function	Calculate outlet fluid temperature [°C]		
	Return	Outlet fluid temperature [°C]		
	Arg.1		Arg.2	-

Figure 6.21 shows the methods related to *Tube* class defined in the Wall class. A *Tube* object can be installed to wall with *AddTube* method. The heat transfer from the wall to the tube [W] can be obtained by *GetHeatTransferToTube* method.

Table 6.21 The methods defined in the Wall class (related to Tube class)

Name	Description			
AddTube	Function	Install the tube to the wall layer.		
	Return	-		
	Arg.1	Tube (Tube type)	Arg.2	Index of the wall layer
RemoveTube	Function	Remove the tube from the wall layer		
	Return	-		
	Arg.1	Index of the wall layer	Arg.2	-
GetHeatTransferToTube	Function	Calculate heat transfer [W] to the tube		
	Return	heat transfer [W] to the tube		
	Arg.1	Index of the wall layer where the tube is installed	Arg.2	-

Figure 6.17 shows a sample program which calculates unsteady heat conduction of wall with heating and cooling tube. Figure 6.16 shows the compositions of the floor. A packed water is installed to raise the heat capacity of the floor.

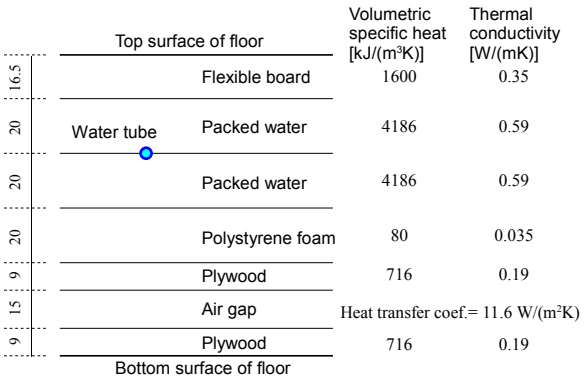


Figure 6.16 Compositions of the floor to calculate

The material of the tube is cross-linked polyethylene (Heat conductance is 0.47 W/(m K)). The span, external diameter and internal diameter is 267mm, 3.4mm and 2.3mm respectively. The flow rate of the fluid is 0.1 L/min/tube. The surface area of floor is 6.48 m^2. The fin efficiency can be calculated as below.

$C_f = 1 / (0.02 / 0.59) + 1 / (0.02 / 0.59) = 59$ W/(m^2 K)

$Z = (0.0267 - 0.0034) \times (59 / (0.59 \times 0.0034))^{0.5} = 3.99$

$\eta_p = 1 / 0.0267 \times (0.0034 + (0.0267\text{-}0.0037)) \times \tanh(3.99) / 3.99 = 0.346$

heat exchanger effectiveness[-] between temperature of fluid and temperature of wall layer can be calculated as below.

$v = (0.01 / 60 / 1000) / (3.14 \times (0.0034 / 2)^2) = 0.0184$ m/s

$\alpha_w = 10.57 \times (1.352 + 0.0198 \times 25) \times 0.0184^{0.8} \times 0.0034^{0.2} = 235.2$

$R_b = (0.0034 - 0.0023) / (3.14 \times 0.47 \times 0.0034) = 0.22$ m K / W

$K_p = 1 / (6.48 / 194.4 \times (1 / (3.14 \times 0.0034 \times 235.2) + 0.22)) = 48.6$ W/(m^2 K)

$\varepsilon_{px} = 1 - \exp(-48.6 \times 6.48 / 4186 / (0.01 \times 54 / 60)) = 0.999$

$\varepsilon_{PNL} = 0.999 / (1 + 0.999 \times 4186 \times (0.01 \times 54 / 60) / 6.48 / 59) \times 1 / 0.34 - 1) = 0.84$

A instance of *Wall* class is created in the line 4~16. To see detailed temperature change, time step is set to 300 seconds. A instance of the *Tube* class is created in the line 18. The fin efficiency and the heat exchanger effectiveness is calculated as above. The *Tube* object is installed in the line 20. The flow rate of the fluid is set to 0 at first. After 50 times iteration, the flow rate is set to 0.54 L/min. Inlet temperature is set to 30 °C constant.

Figure 6.18 shows the result of the simulation. It seems that the wall temperature approaches to outdoor temperature in first 4 hours, since there are no heating by tube. After the supply of 30 °C hot water, wall temperatures rise. Temperature of the neighboring wall layer 2 and 3 rises quickly, and then temperatures of other layers rise slowly.

```
1  /// <summary>Sample program calculating the unsteady heat conduction of wall with heating tube</summary>
2      private static void wallTest2()
3      {
4          WallLayers wl = new WallLayers();
5          wl.AddLayer(new WallLayers.Layer(new WallMaterial(WallMaterial.PredefinedMaterials.FrexibleBoard), 0.0165));
6          wl.AddLayer(new WallLayers.Layer(new WallMaterial("Water", 0.59, 4186), 0.02));
7          wl.AddLayer(new WallLayers.Layer(new WallMaterial("Water", 0.59, 4186), 0.02));
8          wl.AddLayer(new WallLayers.Layer(new WallMaterial(WallMaterial.PredefinedMaterials.ExtrudedPolystyreneFoam_3), 0.02));
9          wl.AddLayer(new WallLayers.Layer(new WallMaterial(WallMaterial.PredefinedMaterials.Plywood), 0.009));
10         wl.AddLayer(new WallLayers.Layer(new WallMaterial(WallMaterial.PredefinedMaterials.AirGap), 0.015));
11         wl.AddLayer(new WallLayers.Layer(new WallMaterial(WallMaterial.PredefinedMaterials.Plywood), 0.009));
12         Wall wall = new Wall(wl);
13         wall.TimeStep = 300;
14         wall.AirTemperature1 = 20;
15         wall.AirTemperature2 = 10;
16         wall.SurfaceArea = 6.48;
17
18         Tube tube = new Tube(0.84, 0.346, 4186);
19         //installing tube to wall
20         wall.AddTube(tube, 1);
21         tube.SetFlowRate(0);  //initial flow rate is 0 kg/s
22         tube.FluidTemperature = 30;
23
24         wall.InitializeTemperature(20); //initialize temperature of the wall
25
26         for (int i = 0; i < wall.Layers.LayerNumber; i++) Console.Write("temperature" + i + ", ");
27         Console.WriteLine("heat transfer to the tube[W], outlet temperature of fluid[C]");
28         for (int i = 0; i < 100; i++)
29         {
30             if (i == 50) tube.SetFlowRate(0.54);  //start heating
31             wall.Update();
32             double[] tmp = wall.GetTemperatures();
33             for (int j = 0; j < tmp.Length - 1; j++) Console.Write(((tmp[j] + tmp[j + 1]) / 2d).ToString("F1") + ", ");
34             Console.Write(wall.GetHeatTransferToTube(1).ToString("F0") + ", " + tube.GetOutletFluidTemperature().ToString("F1"));
35             Console.WriteLine();
36         }
37         Console.Read();
38     }
```

Figure 6.17 The sample program which calculates the unsteady heat conduction of wall with heating tube

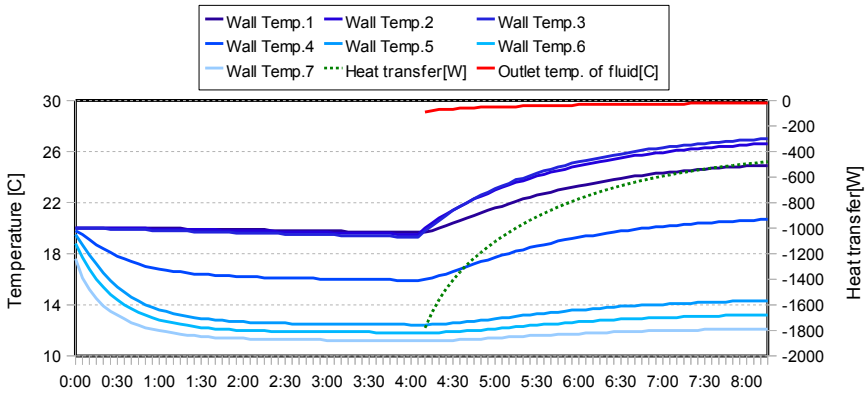


Figure 6.18 Temperature distribution of inner wall (Heating is started at 4:00)

3) Unsteady heat conduction of wall with latent heat storage materials

A phase change material (PCM) is a material with a high heat of fusion. It can store and release large amounts of energy by melting and solidifying. Special setting is needed to treat PCM in *Wall* class, since the heat conductance [W/(m K)] and volumetric specific heat [J/(m^3 K)] is changed when the phase changed.

The *LatentHeatStorageMaterial* class is a model of a PCM. A PCM can be considered as a number of normal materials, since it has different heat conductance [W/(m K)] and volumetric specific heat [J/(m^3 K)] at different state (Figure 6.19).

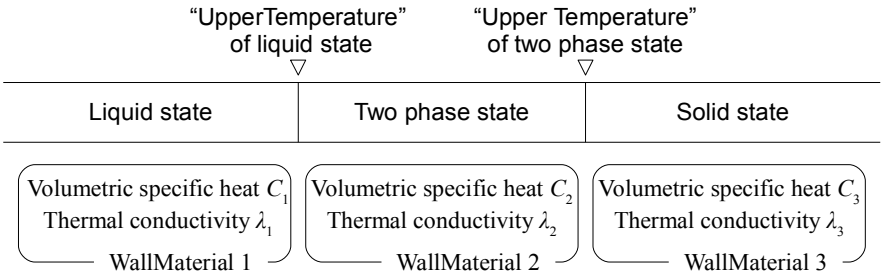


Figure 6.19 Setting normal WallMaterial objects to different state of PCM

Below is the constructor of the *LatentHeatStorageMaterial* class. The second argument is a *WallMaterial* object. The thermodynamic property of the object is applied if the temperature of the latent heat storage material is smaller than the upper temperature limit which is specified by the first argument.

LatentHeatStorageMaterial lmat = new LatentHeatStorageMaterial(upperTemperature, material);

Table 6.22 shows methods defined in the *LatentHeatStorageMaterial* class. User should set a *WallMaterial* objects to different state of a *LatentHeatStorageMaterial* object with *AddMaterial* method.

Table 6.22 The methods defined in the LatentHeatStorageMaterial class

Name	Description			
AddMaterial	Function	Add wall material		
	Return	-		
	Arg.1	upper limit of temperature [°C]	Arg.2	wall material
GetMaterial	Function	Get thermodynamic property of material at given temperature		
	Return	Thermodynamic property of material at given temperature (ImmutableWallMaterial type)		
	Arg.1	Temperature [°C]	Arg.2	-
Initialize	Function	Initialize material temperature		
	Return	-		
	Arg.1	Temperature [°C]	Arg.2	-
GetHeatStorage	Function	Calculate the heat storage [kJ/m^3] from temperature 1 to temperature 2		
	Return	heat storage [kJ/m^3] from temperature 1 to temperature 2		
	Arg.1	Temperature 1 [°C]	Arg.2	Temperature 2 [°C]

Table 6.23 shows the methods related to *LatentHeatStorageMaterial* class defined in *Wall* class.

Table 6.23 The methods defined in Wall class (related to LatentHeatStorageMaterial class)

Name	Description			
SetLatentHeat StorageMaterial	Function	Set latent heat storage material to wall layer.		
	Return	-		
	Arg.1	Index number of the wall layer.	Arg.2	latent heat storage material (LatentHeatStorageMaterial type)
RemoveLatentHeat StorageMaterial	Function	Remove latent heat storage material from wall layer.		
	Return	-		
	Arg.1	Index number of the wall layer.	Arg.2	-

Figure 6.21 shows a sample program which calculates the unsteady heat conduction of wall with latent heat storage material. The composition of the wall is almost same as the wall described at figure 6.16, but in this case, packed water is replaced by latent heat storage material. Table 6.24 shows the thermodynamic properties of the latent heat storage materials.

Table 6.24 Thermodynamic properties of the latent heat storage materials

Layer number	Temperature [°C]	State	Volumetric Specific Heat [kJ/(m^3-K)]	Thermal conductivity [W/(m K)]
Second layer (PCM 1)	~19	solid	5040	0.19
	19~23	two phase	21140	0.205
	23~	liquid	5040	0.22
Third layer (PCM 2)	~30	solid	5004	0.19
	30~32	two phase	88550	0.205
	32~	liquid	4935	0.22

An instance of *LatentHeatStorageMaterial* class is created and set to *Wall* object in the line 24~40. The wall is cooled at first 100 time step. Then, it is heated at next 100 time step. At each time step, wall temperatures and heat storage is written to standard output stream.

Figure 6.20 shows the result of the simulation. The dashed line is temperature of the PCM. The double-dashed line is heat storage of wall. It seems that temperature changes non linearly around 30~32 °C, since the latent heat storage material (PCM 2) has changed it phase from liquid to solid.

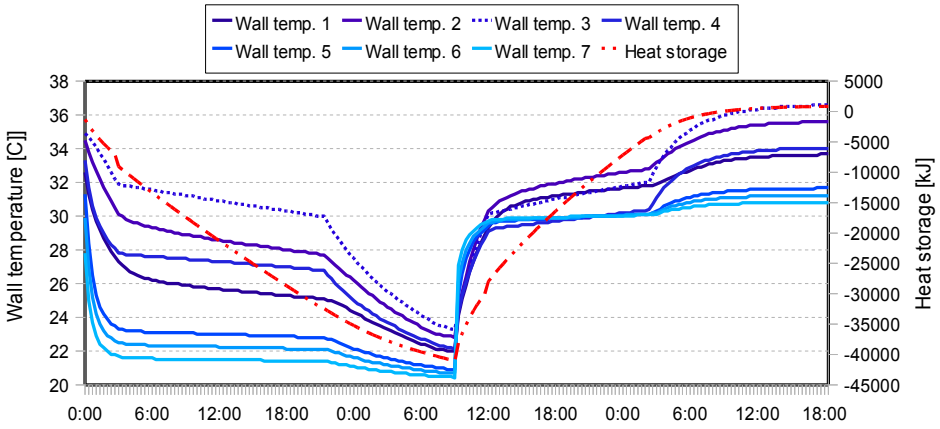


Figure 6.20 The result of the simulation

```
/// <summary>Sample program calculating the unsteady heat conduction of wall with latent heat storage material</summary>
    private static void wallTest3()
    {
        //Initial temperature
        const double INIT_TEMP = 35;

        //Create an instance of WallLayers class
        WallLayers wl = new WallLayers();
        wl.AddLayer(new WallLayers.Layer(new WallMaterial(WallMaterial.PredefinedMaterials.FrexibleBoard), 0.0165));
        wl.AddLayer(new WallLayers.Layer(new WallMaterial("dummy", 1, 1), 0.02));
        wl.AddLayer(new WallLayers.Layer(new WallMaterial("dummy", 1, 1), 0.02));
        wl.AddLayer(new WallLayers.Layer(new WallMaterial(WallMaterial.PredefinedMaterials.ExtrudedPolystyreneFoam_3), 0.02));
        wl.AddLayer(new WallLayers.Layer(new WallMaterial(WallMaterial.PredefinedMaterials.Plywood), 0.009));
        wl.AddLayer(new WallLayers.Layer(new WallMaterial(WallMaterial.PredefinedMaterials.AirGap), 0.015));
        wl.AddLayer(new WallLayers.Layer(new WallMaterial(WallMaterial.PredefinedMaterials.Plywood), 0.009));

        //Create an instance of Wall class
        Wall wall = new Wall(wl);
        wall.TimeStep = 1200;
        wall.AirTemperature1 = 20;
        wall.AirTemperature2 = 20;
        wall.SurfaceArea = 6.48;

        //Create an instance of LatentHeatStorageMaterial class
        LatentHeatStorageMaterial pmc1;
        pmc1 = new LatentHeatStorageMaterial(19, new WallMaterial("PCM1 (Solid)", 0.19, 3.6 * 1400));
        pmc1.AddMaterial(23, new WallMaterial("PCM1 (Two phase)", (0.19 + 0.22) / 2d, 15.1 * 1400));
        pmc1.AddMaterial(100, new WallMaterial("PCM1 (Liquid)", 0.22, 3.6 * 1400));
        pmc1.Initialize(INIT_TEMP);
        //Set PCM to second wall layer
        wall.SetLatentHeatStorageMaterial(1, pmc1);

        //Create an instance of LatentHeatStorageMaterial class
        LatentHeatStorageMaterial pcm2;
        pcm2 = new LatentHeatStorageMaterial(30, new WallMaterial("PCM2 (Solid)", 0.19, 3.6 * 1390));
        pcm2.AddMaterial(32, new WallMaterial("PCM2 (Two phase)", (0.19 + 0.22) / 2d, 63.25 * 1400));
        pcm2.AddMaterial(100, new WallMaterial("PCM2 (Liquid)", 0.22, 3.5 * 1410));
        pcm2.Initialize(INIT_TEMP);
        //Set PCM to third wall layer
        wall.SetLatentHeatStorageMaterial(2, pcm2);

        //Install heating tube between PMCs
        Tube tube = new Tube(0.84, 0.346, 4186);
        wall.AddTube(tube, 1);
        tube.SetFlowRate(0);
        tube.FluidTemperature = 40;

        //Initialize wall temperature
        wall.InitializeTemperature(INIT_TEMP);

        for (int i = 0; i < wall.Layers.LayerNumber; i++) Console.Write("Temperature" + i + ", ");
        Console.WriteLine("Heat storage[kJ]");
        for (int i = 0; i < 200; i++)
        {
            if (i == 100)
            {
                tube.SetFlowRate(0.54); //Start heating
                wall.AirTemperature1 = 30;
                wall.AirTemperature2 = 30;
            }
            wall.Update();
            double[] tmp = wall.GetTemperatures();
            for (int j = 0; j < tmp.Length - 1; j++) Console.Write(((tmp[j] + tmp[j + 1]) / 2d).ToString("F1") + ", ");
            Console.Write(wall.GetHeatStorage(INIT_TEMP).ToString("F0"));
            Console.WriteLine();
        }
        Console.Read();
    }
```

Figure 6.21 The sample program calculating the unsteady heat conduction of wall with latent heat storage material

6.2.7 The classes calculating thermal load of a building

Some classes (*Zone*, *Room*, *MultiRoom* and *Outdoor* class) and interfaces (*IHeatGain* and *ISurface* interface) are defined in this library to calculate the thermal load of a building. Figure 6.22 shows a concept of these classes and interfaces.

The *Room* class is a model of a room which is surrounded by wall and window surfaces. A room can be separated into some smaller zones which have different characteristics from the other zone. These zones can be modeled by the *Zone* class. The air state of a zone is expressed by a single *MoistAir* object and the vertical or horizontal distribution of the air is not considered. The heat production in a zone is modeled by the *IHeatGain* interface. User can make their original heat gain class by implementing the *IHeatGain* interface. *ISurface* interface is a model of a wall surface or window surface. The opposite side of a surface is either outdoor or other rooms. The outdoor weather state is modeled by the Outdoor class.

The state of the room temperature and the humidity can be updated by the *Update* methods defined in the *Zone* class and the *MultiRoom* class. The *Update* method defined in the *Zone* class uses simpler solution method than that defined in the *MultiRoom* class. The temperature of the room is affected by temperature of surrounding wall surfaces. These surface temperatures are affected by the temperature of opposite side room. If there is room to room ventilation, the temperature of the room mutually affect each other directly. Therefore, the temperature of the rooms should be solved simultaneously to be exact. The *Update* methods defined in the *Zone* class uses simple model which assume that the temperatures of the neighboring rooms are same as last time step. The *Update* method defined in the *MultiRoom* class uses detailed model which solves all the temperatures of the rooms simultaneously. Although this model needs relatively long calculation time than the previous one. Usage of the *Update* methods are explained later with concrete example of a building.

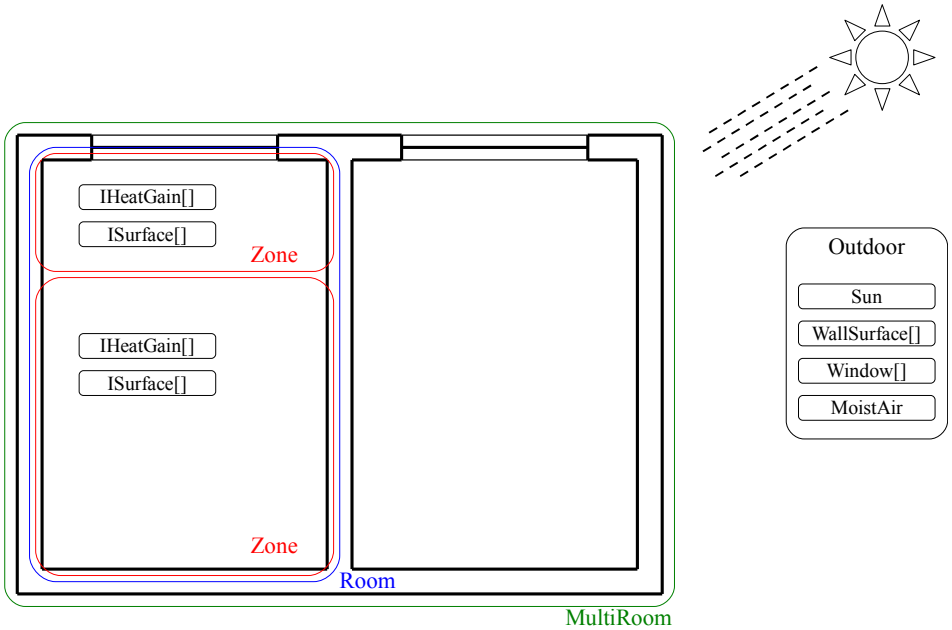


Figure 6.22 classes and interfaces calculating thermal load of a building

1) Description of a sample building to be modeled

Figure 6.23 shows the plan and the section of the sample building. The building has east and west room which has a south-facing window. The window of the east room has a sun shade whose width and overhang is 5m and 1m respectively. The sun shade is set 0.5m above from the window. The depth of the room is 7m. We treat area 3m from the window as perimeter zone. The ventilation volume of the west room is 10 CMH. Outdoor air is taken in to the room from the perimeter zone and exhausted from the interior zone. There is no air-intake from outdoor at the east zone. There is 10 CMH interior ventilation between the perimeter and interior zone at the east zone. A heat production element is set in the east interior zone (Sensible radiative heat gain=100W, Sensible convective heat gain=100W, Latent heat gain=20W). The glasses of the windows are low-emissivity coating single glass. It's width and height is 3m and 2m. All the walls are concrete and thickness is 400 mm. The building has no substructure and directly set on the ground. The temperature of the ground is treated as constant (28°C).

Table 6.25, figure 6.24 and figure 6.25 shows weather data. It is data measured in Tokyo Japan in 2007/8/3.

The rooms are maintained at constant temperature (26°C) and humidity (0.01 kg/kg(DA)) from 8:00 ~ 19:00.

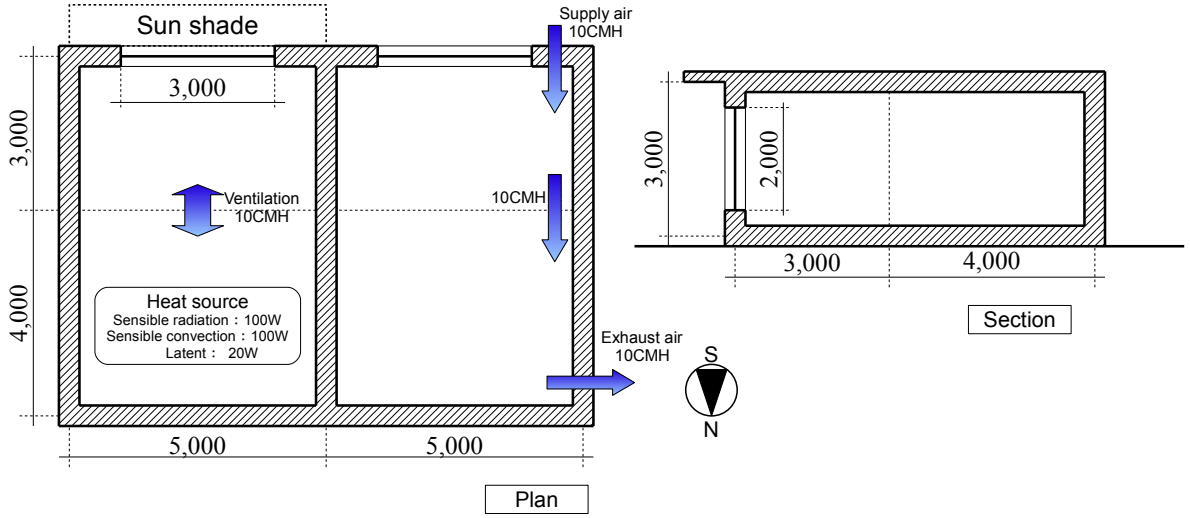


Figure 6.23 The plan and the section of the building

Table 6.25 Sample weather data (Tokyo : 2007/8/3)

Hour	Drybulb temperature [℃]	Humidity ratio [kg/kg(DA)]	Nocturnal Radiation [W/m²]	Direct normal radiation [W/m²]	Diffuse horizontal radiation [W/m²]
0:00	24.2	0.0134	32	0	0
1:00	24.1	0.0136	30	0	0
2:00	24.1	0.0134	30	0	0
3:00	24.2	0.0133	29	0	0
4:00	24.3	0.0131	26	0	0
5:00	24.2	0.0134	24	0	0
6:00	24.4	0.0138	24	106	36
7:00	25.1	0.0142	25	185	115
8:00	26.1	0.0142	25	202	198
9:00	27.1	0.0140	25	369	259
10:00	28.8	0.0147	24	427	314
11:00	29.9	0.0149	24	499	340
12:00	30.7	0.0142	24	557	340
13:00	31.2	0.0146	23	522	349
14:00	31.6	0.0140	24	517	319
15:00	31.4	0.0145	24	480	277
16:00	31.3	0.0144	24	398	228
17:00	30.8	0.0146	24	255	167
18:00	29.4	0.0142	23	142	87
19:00	28.1	0.0136	23	2	16
20:00	27.5	0.0136	24	0	0
21:00	27.1	0.0135	26	0	0
22:00	26.6	0.0136	25	0	0
23:00	26.3	0.0140	23	0	0

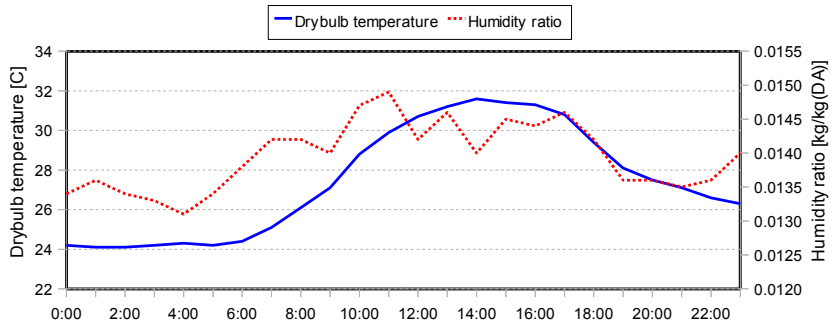


Figure 6.24 Weather data (Drybulb temperature and Humidity ratio)

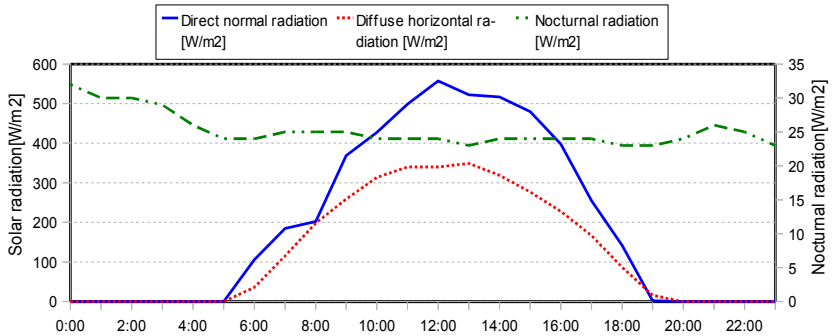


Figure 6.25 Weather data (Solar radiation and nocturnal radiation)

2) Simulate with Zone class (Simplified algorithm)

The *Zone* class solves equation 6.18~6.21 to determine the temperature, humidity ratio, sensible heat load and latent heat load [2) 18)]. The convective heat transfer between room air and surfaces is treated simply in this mode. As shown in equation 6.18, temperatures (T_{MRT}) and convective heat transfer coefficients ($\alpha_{(i)}$ k_c) are treated as same value for the all surfaces

$$ZN_S \frac{d T_R}{dt} = \sum_{n=1}^{NW} A_n \alpha_{(i)} k_c (T_{MRT} - T_R) + c_a G_o (T_a - T_R) + HG_{(c)} - HE_s \tag{6.18}$$

$$ZN_S = c_a \rho \cdot VOL + CPF \tag{6.19}$$

$$ZN_L \frac{d x_R}{dt} = G_o (x_a - x_R) + LG - LE \quad \text{†1)} \tag{6.20}$$

$$ZN_L = \rho \cdot VOL + LCPF \tag{6.21}$$

ZN_S : sensible heat capacity of the zone [J/K], T_R : drybulb temperature of the zone [°C], t : time [sec], NW : number of surfaces belong to zone
A_n : area of surface [m^2], $\alpha_{(i)}$: film coefficient [W/(m^2 K)], k_c : convective rate of film coefficient [-]
T_{MRT} : mean radiant temperature of surfaces [°C], c_a : specific heat of the air [J/(kg K)], G_o : ventilation [kg/s]
T_a : drybulb temperature of ventilation air [°C], $HG_{(c)}$: sensible heat production in the zone (convective) [W]
HE_s : sensible heat supply to the zone [W], ρ : density of the air [kg/m^3], VOL : zone volume [m^3]
CPF : additional sensible heat capacity of the zone (furniture) [J/K], ZN_L : latent heat capacity of the zone [kg]
x_R : humidity ratio of zone air [kg/kg(DA)], x_a : humidity ratio of ventilation air [kg/kg(DA)]
LG : latent heat production in the zone [kg/s], LE : latent heat supply to the zone [kg/s]
$LCPF$: additional latent heat supply to the zone (furniture) [kg]

Table 6.26 and 6.27 shows the properties defined in the *Zone* class. *SensibleHeatCapacity* is sensible heat capacity of the zone (*CPF* in the equation 6.19). It is mainly heat capacity of furniture and doesn't include the heat capacity of the room air and building structure. It is 12~15 kJ/(m^3 K) for the office building [19) 20)].

LatentHeatCapacity is latent heat capacity of the zone (*LCPF* in the equation 6.21). As same as *SensibleHeatCapacity* property, it doesn't include latent heat capacity of the room air. It can be set to 0 for the most cases [†2)].

FilmCoefficient is film coefficients of the surfaces ($\alpha_{(i)}$ in the equation 6.18). The *Zone* class assumes that the film coefficient of all surfaces are the same value. k_c in the equation 6.18 is set by *SetConvectiveRate*.

To calculate drybulb temperature or humidity ratio, solve equations for T_r or x_R. To calculate sensible or latent heat load, solve equations for HE_s or LE. The equations are solved for T_r or x_R when the *DrybulbTemperatureSetPoint* or *AbsoluteHumiditySetPoint* properties are set to true. If these properties are set to false, free float temperature and humidity ratio is calculated.

AtmosphericPressure is atmospheric pressure and it has an effect to density of the room air. *Surfaces* and *HeatGains* are the list of surfaces and heat production elements in the room. These elements can be set by *AddSurface* method or *AddHeatGain* method.

†1) 参考文献 2)の宇田川の式に水蒸気容量の項を加えた
†2) 松尾陽 ：室の潜熱容量について, 日本建築学会大会学術講演梗概集, 1987, 等を参照して下さい。

Table 6.26 The properties defined in the Zone class

Name	Description	Has set accessor	Unit	Type
TimeStep	Time step of calculation	TRUE	sec	double
Name	Name	TRUE	-	string
Volume	Volume	TRUE	m^3	double
SensibleHeatCapacity	Sensible heat capacity It doesn't include heat cap. of air.	TRUE	J/K	double
SensibleHeatSupply	Sensible heat supply Heating : >0, Cooling : < 0	TRUE	W	double
LatentHeatCapacity	Latent heat capacity It doesn't include heat cap. of air.	TRUE	kg	double
LatentHeatSupply	Latent heat supply humidify : >0, Dehumidify : < 0	TRUE	W	double
FilmCoefficient	Film coefficient of the surfaces	TRUE	$W/(m^2 K)$	double
VentilationVolume	Vintilation volume	TRUE	m^3/h	double
VentilationAirState	Ventilation air state	TRUE	-	MoistAir
ControlDrybulbTemperature	Control drybulb temperature or not	TRUE	-	bool
ControlHumidityRatio	Control humidity ratio or not	TRUE	-	bool
DrybulbTemperatureSetPoint	Set point of drybulb temperature	TRUE	°C	double
HumidityRatioSetPoint	Set point of humidity ratio	TRUE	kg/kg(DA)	dobule
Surfaces	List of surfaces belonging to room	-	-	ImmutableSurface[ ]
HeatGains	List of heat gains	-	-	IheatGain[ ]
AtmosphericPressure	Atmospheric pressure	TRUE	kPa	double
CurrentDrybulbTemperature	Current drybulb temperature	-	°C	double
CurrentHumidityRatio	Current humidity raio	-	kg/kg(DA)	double
CurrentSensibleHeatLoad	Current sensible heat load	-	W	double
CurrentLatentHeatLoad	Current latent heat load	-	W	double
CurrentMeanRadiantTemperature	Current mean radiant temperature	-	°C	double
CurrentDateTime	Current date and time	TRUE	-	DateTime

Table 6.27 The methods defined in the Zone class (Related to setting boundary conditions)

Name	Description			
AddSurface	Function	Add a surface to the zone		
	Return	Successfully added or not		
	Arg.1	Surface to add. (ISurface object)	Arg.2	-
RemoveSurface	Function	Remove a surface from the zone		
	Return	Successfully removed or not		
	Arg.1	Surface to remove. (ISurface object)	Arg.2	-
AddWindow	Function	Add a window to the zone		
	Return	Successfully added or not		
	Arg.1	Window to add (Window object)	Arg.2	-
RemoveWindow	Function	Remove a window from the zone		
	Return	Successfully removed or not.		
	Arg.1	Window to remove. (Window object)	Arg.2	-
AddHeatGain	Function	Add heat producing element to the zone		
	Return	Successfully added or not.		
	Arg.1	Heat producing element to add (IHeatGain object)	Arg.2	-
RemoveHeatGain	Function	Remove heat producing element from the zone		
	Return	Successfully removed or not		
	Arg.1	Heat producing element to remove (IHeatGain object)	Arg.2	-
SetConvectiveRate	Function	Set convective rate [-] of film coefficient [W/(m^2 K)]		
	Return	-		
	Arg.1	Convective rate [-]	Arg.2	-
InitializeAirState	Function	Initialize air state of the zone		
	Return	-		
	Arg.1	Drybulb temperature [°C]	Arg.2	Humidity ratio [kg/kg(DA)]

Table 6.28 shows the methods defined in the *Zone* class. The drybulb temperature, humidity ratio, sensible heat load and latent heat load of the zone is updated by *Update* method. The state of the zone at next time step can be calculated by the methods named as *GetNext~~~*. They don't update the states of the zone.

Table 6.28 The methods defined in the Zone class (Related to updating the model)

Name	Description			
Update	Function	Update the drybulb temperature, humidity ratio, sensible heat load and latent heat load of the zone		
	Return	-		
	Arg.1	-	Arg.2	-
GetNextDrybulbTemperature	Function	Calculate the drybulb temperature [°C] with the sensible heat supply [W] indicated.		
	Return	Drybulb temperature [°C] of the zone		
	Arg.1	Sensible heat supply [W] (Treat heating as positive value)	Arg.2	-
GetNextHumidityRatio	Function	Calculate the humidity ratio [kg/kg(DA)] with the latent supply [W] indicated.		
	Return	Humidity ratio [kg/kg(DA)]		
	Arg.1	Latent heat supply[W] (Treat humidifying as positive value)	Arg.2	-
GetNextMeanRadiantTemperature	Function	Calculate mean radiant temperature [°C] at next time step.		
	Return	Mean radiant temperature [°C]		
	Arg.1	-	Arg.2	-
GetNextSensibleHeatLoad	Function	Calculate the sensible heat load [W] with the drybulb temperature [°C] indicated.		
	Return	Sensible heat supply [W] (Treat heating as positive value)		
	Arg.1	Drybulb temperature [°C]	Arg.2	-
GetNextLatentHeatLoad	Function	Calculate the latent heat load [W] with the humidity ratio [kg/kg(DA)] indicated.		
	Return	Latent heat load [W] (Treat humidifying as positive value)		
	Arg.1	Humidity ratio [kg/kg(DA)]	Arg.2	-

The sol-air temperature near wall and window surfaces can be represented by equation 6.22. The upper line represents an air at surface facing to the zone. The middle line represents an air at surface facing to the outdoor. The lower line is in the case surface is directly connected to ground. The *Outdoor* class calculates the middle and the lower equation.

Table 6.29 and 6.30 shows the properties and methods defined in the *Outdoor* class. The surface of the walls and windows can be set by the methods named *Add~~* and *Remove~~*. When the *SetWallSurfaceBoundaryState* method is called, sol-air temperatures are set to the surfaces. Outdoor boundary states can be set by *AirState*, *Sun*, *GroundTemperature* and *NocturnalRadiation* properties.

$$T_{SOL}=\begin{cases}\dfrac{a_{(c)}T_R+a_{(r)}T_{MRT}+RS}{a_{(i)}} \\ \dfrac{a_s I_W-\varepsilon F_S RN}{\alpha_o}+T_a \\ T_{GRZ}\end{cases} \tag{6.22}$$

T_{SOL} : sol-air temperature of a surface [°C], T_R : drybulb temperature of a zone [°C], T_{MRT} : mean radiant temperature of surfaces [°C]
RS : radiation to a surface [W/m^2], $\alpha_{(i)}$: film coefficient of surface [W/(m^2 K)], $\alpha_{(c)}$: convective heat transfer coefficient [W/(m^2 K)]
$\alpha_{(r)}$: radiative heat transfer coefficient [W/(m^2 K)], a_s : solar absorptivity [-], I_W : solar incident [W/m^2], ε : emissivity [-]
F_S : form factor from incline to the sky [-], RN : nocturnal radiation [W/m^2], T_a : drybulb temperature of outdoor air [°C]
T_{GRZ} : ground temperature [°C]

Table 6.29 The properties defined in the Outdoor class

Name	Description	Has set accessor	Unit	Type
AirState	Moist air state of outdoor	TRUE	-	MoistAir
Sun	Sun	TRUE	-	Sun
WallSurfaces	List of wall surfaces facing the outdoor	TRUE	-	ImmutableWallSurface []
GroundWallSurfaces	List of wall surfaces contacted to ground	-	-	ImmutableWallSurface []
Windows	List of windows	-	-	ImmutableWindow []
GroundTemperature	Temperature of the ground	TRUE	°C	double
NocturnalRadiation	Nocturnal radiation	TRUE	W/m^2	double

Table 6.30 The methods defined in the Outdoor class

Name	Description			
SetWallSurfaceBoundaryState	Function	Set outdoor state to wall surfaces		
	Return	-		
	Arg.1	-	Arg.2	-
AddWallSurface	Function	Add a wall surface which is exposed to the outdoor		
	Return	Successfully added or not		
	Arg.1	Wall surface to be added (WallSurface type)	Arg.2	-
RemoveWallSurface	Function	Remove a wall surface which is exposed to the outdoor		
	Return	Successfully removed or not		
	Arg.1	Wall surface to be removed (WallSurface type)	Arg.2	-
AddGroundWallSurface	Function	Add a wall surface which is contacted to the ground		
	Return	Successfully added or not		
	Arg.1	Wall surface to be added (WallSurface type)	Arg.2	-
RemoveGroundWallSurface	Function	Remove a wall surface which is contacted to the ground		
	Return	Successfully removed or not		
	Arg.1	Wall surface to be removed (WallSurface type)	Arg.2	-
AddWindow	Function	Add a window		
	Return	Successfully added or not		
	Arg.1	Window to be added (Window type)	Arg.2	-
RemoveWindow	Function	Remove a window		
	Return	Successfully removed or not		
	Arg.1	Window to be removed (Window type)	Arg.2	-
GetRadiationToIncline	Function	Calculate incoming radiation [W/m^2] to given Incline object		
	Return	Incoming radiation [W/m^2] to given Incline object		
	Arg.1	Incline (Incline type)	Arg.2	Albedo [-]
	Arg.3	Shadow rate [-]	Arg.4	-
SetConvectiveRate	Function	Set convective rate [-] of the wall surface film coefficient		
	Return	-		
	Arg.1	Convective rate [-]	Arg.2	-
SetFilmCoefficient	Function	Set film coefficient [W/(m^2 K)] to wall surface.		
	Return	-		
	Arg.1	Film coefficient [W/(m^2 K)]	Arg.2	Convective rate [-]

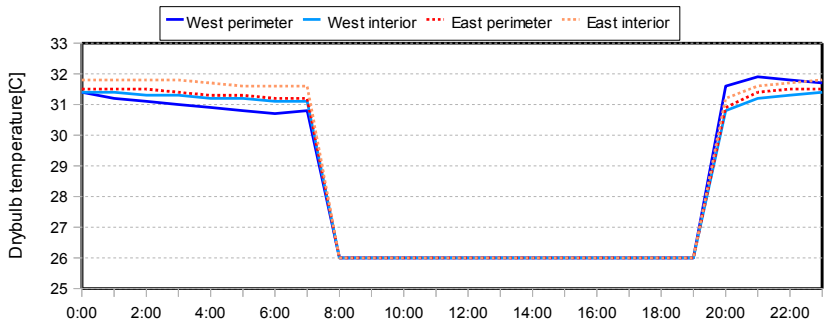
Figure 6.28 shows a sample program which calculates an air state and a heat load of the building discussed at 1).

Below shows brief flow of the program.

1) Create an instance of the *Wall* and the *Window* class.
2) Set the surface of the *Wall* and *Window* object to the *Zone* and *Outdoor* object
3) Update outdoor state.
4) Set sol-air temperature to outdoor facing surfaces by *Outdoor* object.
5) Update *Wall* objects.
6) Update *Zone* objects.
7) Set sol-air temperature to indoor facing surfaces by *Zone* object.
8) Back to 3)

A sample weather data is prepared in the line 4~16. Instances of the *Incline* class are created in the line 24~29. These instances are used to initialize wall and window surfaces. Instances of the *Zone* class are created in the line 31~47. There are four zones (West interior, west perimeter, east interior and east perimeter) in this building. Instances of the Wall class are created in the line 60~163. The surfaces of these *Wall* objects are set to the *Zone* objects and the *Outdoor* object. As same as wall surfaces, window surface is set to the *Zone* objects and the *Outdoor* object in the line 174~185. The states of the zones are updated in the line 193~245. 24 hours calculation is iterated 100 times to get steady state result.

Figure 6.26 and 6.27 shows the results of the simulation. The temperatures of the zones maintained at 26 °C from 8:00 to 19:00 since HVAC system is operating. The heat load of the west perimeter zone is relatively bigger than that of other zones, since the west perimeter intakes air directly from outdoor.



Figre 6.26 Time-series behavior of room temperatures

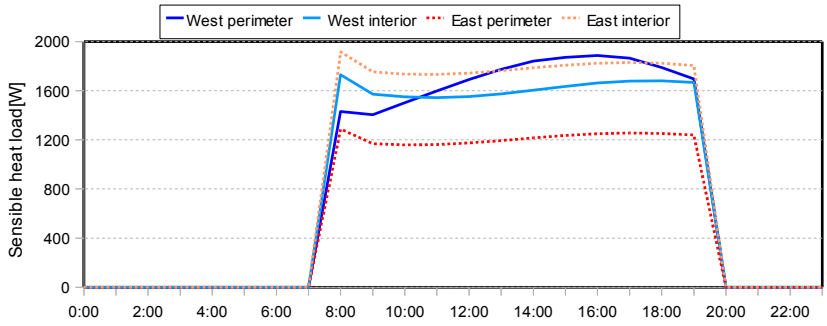


Figure 6.27 Time-series of sensible heat load

```
/// <summary>Sample program calculating the air state and heat load of the building (Zone class)</summary>
private static void AirStateAndHeatLoadTest1()
{
   //A sample weather data
   //Drybulb temperature [C]
   double[] dbt = new double[] { 24.2, 24.1, 24.1, 24.2, 24.3, 24.2, 24.4, 25.1, 26.1, 27.1, 28.8, 29.9,
      30.7, 31.2, 31.6, 31.4, 31.3, 30.8, 29.4, 28.1, 27.5, 27.1, 26.6, 26.3 };
   //Humidity ratio [kg/kg(DA)]
   double[] hum = new double[] { 0.0134, 0.0136, 0.0134, 0.0133, 0.0131, 0.0134, 0.0138, 0.0142, 0.0142, 0.0140, 0.0147, 0.0149,
      0.0142, 0.0146, 0.0140, 0.0145, 0.0144, 0.0146, 0.0142, 0.0136, 0.0136, 0.0135, 0.0136, 0.0140 };
   //Nocturnal radiation [W/m2]
   double[] nrd = new double[] { 32, 30, 30, 29, 26, 24, 24, 25, 25, 25, 24, 24, 24, 23, 24, 24, 24, 24, 23, 23, 24, 26, 25, 23 };
   //Direct normal radiation [W/m2]
   double[] dnr = new double[] { 0, 0, 0, 0, 0, 0, 106, 185, 202, 369, 427, 499, 557, 522, 517, 480, 398, 255, 142, 2, 0, 0, 0, 0 };
   //Diffuse horizontal radiation [W/m2]
   double[] drd = new double[] { 0, 0, 0, 0, 0, 0, 36, 115, 198, 259, 314, 340, 340, 349, 319, 277, 228, 167, 87, 16, 0, 0, 0, 0 };

   //Create an instance of the Outdoor class
   Outdoor outdoor = new Outdoor();
   Sun sun = new Sun(Sun.City.Tokyo);  //Located in Tokyo
   outdoor.Sun = sun;
   outdoor.GroundTemperature = 25;     //Ground temperature is assumed to be constant

   //Create an instance of the Incline class
   Incline nIn = new Incline(Incline.Orientation.N, 0.5 * Math.PI); //North, Vertical
   Incline eIn = new Incline(Incline.Orientation.E, 0.5 * Math.PI); //East, Vertical
   Incline wIn = new Incline(Incline.Orientation.W, 0.5 * Math.PI); //West, Vertical
   Incline sIn = new Incline(Incline.Orientation.S, 0.5 * Math.PI); //South, Vertical
   Incline hIn = new Incline(Incline.Orientation.S, 0);  //Horizontal

   //Create an instance of the Zone class
   Zone[] zones = new Zone[4];
   Zone wpZone = zones[0] = new Zone("West perimeter zone");
   wpZone.Volume = 3 * 5 * 3;  //Ceiling height is 3m
   Zone wiZone = zones[1] = new Zone("West interior zone");
   wiZone.Volume = 4 * 5 * 3;
   Zone epZone = zones[2] = new Zone("East perimeter zone");
   epZone.Volume = 3 * 5 * 3;
   Zone eiZone = zones[3] = new Zone("East interior zone");
   eiZone.Volume = 4 * 5 * 3;
   foreach (Zone zn in zones)
   {
      zn.VentilationVolume = 10; //Ventilation volume[CMH]
      zn.TimeStep = 3600;
      zn.DrybulbTemperatureSetPoint = 26;
      zn.HumidityRatioSetPoint = 0.01;
   }

   //Set a heat production element to the east interior zone
   //Convective sensible heat=100W, Radiative sensible heat=100W, Latent heat=20W
   eiZone.AddHeatGain(new ConstantHeatGain(100, 100, 20));

   //Create an instance of the WallLayers class : Concrete,400mm
   WallLayers wl = new WallLayers();
   wl.AddLayer(new WallLayers.Layer(new WallMaterial(WallMaterial.PredefinedMaterials.ReinforcedConcrete), 0.4));

   //Create an instance of the GlassPanes class:Low-emissivity coating single glass
   GlassPanes gPanes = new GlassPanes(new GlassPanes.Pane(GlassPanes.Pane.PredifinedGlassPane.HeatReflectingGlass06mm));

   //Set wall surfaces to the zone objects
   Wall[] walls = new Wall[18];
   List<WallSurface> outdoorSurfaces = new List<WallSurface>();
   Wall wpwWall = walls[0] = new Wall(wl, "West wall in the west perimeter zone");
   wpwWall.SurfaceArea = 3 * 3;
   outdoorSurfaces.Add(wpwWall.GetSurface(true));
   wpZone.AddSurface(wpwWall.GetSurface(false));
   wpwWall.SetIncline(wIn, true);

   Wall wpcWall = walls[1] = new Wall(wl, "Ceiling in the west perimeter zone");
   wpcWall.SurfaceArea = 3 * 5;
   outdoorSurfaces.Add(wpcWall.GetSurface(true));
   wpZone.AddSurface(wpcWall.GetSurface(false));
   wpcWall.SetIncline(hIn, true);

   Wall wpfWall = walls[2] = new Wall(wl, "Floor in the west perimeter zone");
   wpfWall.SurfaceArea = 3 * 5;
   outdoor.AddGroundWallSurface(wpfWall.GetSurface(true));
   wpZone.AddSurface(wpfWall.GetSurface(false));

   Wall winWall = walls[3] = new Wall(wl, "North wall in the west interior zone");
```

```
    winWall.SurfaceArea = 3 * 5;
    outdoorSurfaces.Add(winWall.GetSurface(true));
    wiZone.AddSurface(winWall.GetSurface(false));
    winWall.SetIncline(nIn, true);

    Wall wiwWall = walls[4] = new Wall(wl, "West wall in the west interior zone");
    wiwWall.SurfaceArea = 3 * 4;
    outdoorSurfaces.Add(wiwWall.GetSurface(true));
    wiZone.AddSurface(wiwWall.GetSurface(false));
    wiwWall.SetIncline(wIn, true);

    Wall wicWall = walls[5] = new Wall(wl, "Ceiling in the west interior zone");
    wicWall.SurfaceArea = 4 * 5;
    outdoorSurfaces.Add(wicWall.GetSurface(true));
    wiZone.AddSurface(wicWall.GetSurface(false));
    wicWall.SetIncline(hIn, true);

    Wall wifWall = walls[6] = new Wall(wl, "Floor in the west interior zone");
    wifWall.SurfaceArea = 4 * 5;
    outdoor.AddGroundWallSurface(wifWall.GetSurface(true));
    wiZone.AddSurface(wifWall.GetSurface(false));

    Wall epwWall = walls[7] = new Wall(wl, "East wall in the east perimeter zone");
    epwWall.SurfaceArea = 3 * 3;
    outdoorSurfaces.Add(epwWall.GetSurface(true));
    epZone.AddSurface(epwWall.GetSurface(false));
    epwWall.SetIncline(eIn, true);

    Wall epcWall = walls[8] = new Wall(wl, "Ceiling in the east perimeter zone");
    epcWall.SurfaceArea = 3 * 5;
    outdoorSurfaces.Add(epcWall.GetSurface(true));
    epZone.AddSurface(epcWall.GetSurface(false));
    epcWall.SetIncline(hIn, true);

    Wall epfWall = walls[9] = new Wall(wl, "Floor in the east perimeter zone");
    epfWall.SurfaceArea = 3 * 5;
    outdoor.AddGroundWallSurface(epfWall.GetSurface(true));
    epZone.AddSurface(epfWall.GetSurface(false));

    Wall einWall = walls[10] = new Wall(wl, "North wall in the east interior zone");
    einWall.SurfaceArea = 5 * 3;
    outdoorSurfaces.Add(einWall.GetSurface(true));
    eiZone.AddSurface(einWall.GetSurface(false));
    einWall.SetIncline(nIn, true);

    Wall eiwWall = walls[11] = new Wall(wl, "East wall in the east interior zone");
    eiwWall.SurfaceArea = 4 * 3;
    outdoorSurfaces.Add(eiwWall.GetSurface(true));
    eiZone.AddSurface(eiwWall.GetSurface(false));
    eiwWall.SetIncline(eIn, true);

    Wall eicWall = walls[12] = new Wall(wl, "Ceiling in the east interior zone");
    eicWall.SurfaceArea = 4 * 5;
    outdoorSurfaces.Add(eicWall.GetSurface(true));
    eiZone.AddSurface(eicWall.GetSurface(false));
    eicWall.SetIncline(hIn, true);

    Wall eifWall = walls[13] = new Wall(wl, "Floor in the east interior zone");
    eifWall.SurfaceArea = 4 * 5;
    outdoor.AddGroundWallSurface(eifWall.GetSurface(true));
    eiZone.AddSurface(eifWall.GetSurface(false));

    Wall cpWall = walls[14] = new Wall(wl, "Inner wall at perimeter");
    cpWall.SurfaceArea = 3 * 3;
    wpZone.AddSurface(cpWall.GetSurface(true));
    epZone.AddSurface(cpWall.GetSurface(false));

    Wall ciWall = walls[15] = new Wall(wl, "Inner wall at interior");
    ciWall.SurfaceArea = 4 * 3;
    wiZone.AddSurface(ciWall.GetSurface(true));
    eiZone.AddSurface(ciWall.GetSurface(false));

    Wall wpsWall = walls[16] = new Wall(wl, "South wall in the west perimeter zone");
    wpsWall.SurfaceArea = 5 * 3 - 3 * 2;    //Reduce window surface area
    outdoorSurfaces.Add(wpsWall.GetSurface(true));
    wpZone.AddSurface(wpsWall.GetSurface(false));
    wpsWall.SetIncline(sIn, true);

    Wall epsWall = walls[17] = new Wall(wl, "South wall in the east perimeter zone");
    epsWall.SurfaceArea = 5 * 3 - 3 * 2;    //Reduce window surface area
    outdoorSurfaces.Add(epsWall.GetSurface(true));
```

```
    epZone.AddSurface(epsWall.GetSurface(false));
    epsWall.SetIncline(sIn, true);

    //Initialize outdoor surfaces
    foreach (WallSurface ws in outdoorSurfaces)
    {
        //Add wall surfaces to Outdoor object
        outdoor.AddWallSurface(ws);
        //Initialize emissivity of surface
        ws.InitializeEmissivity(WallSurface.SurfaceMaterial.Concrete);
    }

    //Add windows to the west zone
    Window wWind = new Window(gPanes, "Window in the west perimeter zone");
    wWind.SurfaceArea = 3 * 2;
    wpZone.AddWindow(wWind);
    outdoor.AddWindow(wWind);
    //Add windows to the east zone
    Window eWind = new Window(gPanes, "Window in the east perimeter zone");
    eWind.SurfaceArea = 3 * 2;
    //Set horizontal sun shade.
    eWind.Shade = SunShade.MakeHorizontalSunShade(3, 2, 1, 1, 1, 0.5, sIn);
    wpZone.AddWindow(eWind);
    outdoor.AddWindow(eWind);

    //Output title wrine to standard output stream
    StreamWriter sWriter = new StreamWriter("AirStateAndHeatLoadTest1.csv");
    foreach (Zone zn in zones) sWriter.Write(zn.Name + "Drybulb temperature[C], " + zn.Name +
                "Humidity ratio[kg/kgDA], " + zn.Name + "Sensible heat load[W], " + zn.Name + "Latent heat load[W], ");
    sWriter.WriteLine();

    //Update the state (Iterate 100 times to make state steady)
    for (int i = 0; i < 100; i++)
    {
        DateTime dTime = new DateTime(2007, 8, 3, 0, 0, 0);
        for (int j = 0; j < 24; j++)
        {
            //Set date and time to Sun and Zone object.
            sun.Update(dTime);
            foreach (Zone zn in zones) zn.CurrentDateTime = dTime;

            //Operate HVAC system (8:00~19:00)
            bool operating = (8 <= dTime.Hour && dTime.Hour <= 19);
            foreach (Zone zn in zones)
            {
                zn.ControlHumidityRatio = operating;
                zn.ControlDrybulbTemperature = operating;
            }

            //Set weather state.
            outdoor.AirState = new MoistAir(dbt[j], hum[j]);
            outdoor.NocturnalRadiation = nrd[j];
            sun.SetGlobalHorizontalRadiation(drd[j], dnr[j]);

            //Set ventilation air state.
            eiZone.VentilationAirState = new MoistAir(epZone.CurrentDrybulbTemperature, eiZone.CurrentHumidityRatio);
            epZone.VentilationAirState = new MoistAir(eiZone.CurrentDrybulbTemperature, eiZone.CurrentHumidityRatio);
            wpZone.VentilationAirState = outdoor.AirState;
            wiZone.VentilationAirState = new MoistAir(wpZone.CurrentDrybulbTemperature, wpZone.CurrentHumidityRatio);

            //Update boundary state of outdoor facing surfaces.
            outdoor.SetWallSurfaceBoundaryState();

            //Update the walls.
            foreach (Wall wal in walls) wal.Update();

            //Update the zones.
            foreach (Zone zn in zones) zn.Update();

            //Update date and time
            dTime = dTime.AddHours(1);

            //If it is last iteration, output result to CSV text.
            if (i == 99)
            {
                foreach (Zone zn in zones)
                {
                    sWriter.Write(zn.CurrentDrybulbTemperature.ToString("F1") + ", " + zn.CurrentHumidityRatio.ToString("F3") + ", " +
                    zn.CurrentSensibleHeatLoad.ToString("F0") + ", " + zn.CurrentLatentHeatLoad.ToString("F0") + ", ");
                }
                sWriter.WriteLine();
```

```
243             }
244         }
245     }
246
247     sWriter.Close();
248 }
```

Figure 6.28 The sample program calculating the air state and the heat load with Zone class

3) Simulate with MultiRoom class (Detailed algorithm)

The MutliRoom class can simultaneously solve more than one zone temperatures. As previously shown in the figure 6.22, building is treated as three layers (*Zone*, *Room* and *MultiRoom*) hierarchic structure in this class library. The Zone class defines the extent of convective heat transfer. The Room class defines the extent of radiative heat transfer. Figure 6.29 shows concept of these heat transfers.

As shown on the left of the figure 6.29, the moist air of the zone doesn't exchange heat with surfaces belonging to other zones. In this figure, Z1 only exchange heat with surface W1, W2 and W3. It doesn't exchange heat with W4, W5 and W6. On the other hand, radiative heat transfer occurs not only single zone. As shown on the right of the figure 6.29, the surfaces belonging same room exchanges heat each other. In this figure, the surface W1, W2, W3, W4, W5 and W6 exchanges heat each other.

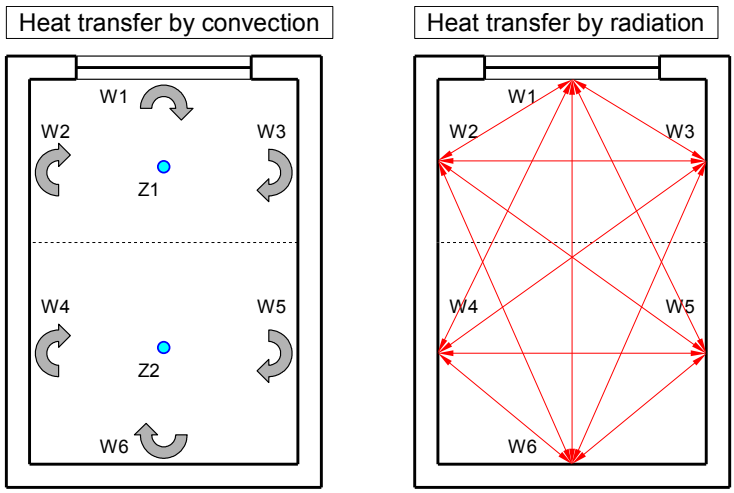


Figure 6.29 Radiative and convective heat transfer in the room

The *MultiRoom* class solves equation 6.23 to determine the room state. Unlike equation 6.18, equation 6.23 evaluate convective heat transfer between wall surface and room individually. The temperatures and the convective heat transfer coefficients of surfaces are treated as different value [†1].

$$ZN_S \frac{d\,T_R}{dt} = \sum_{n=1}^{NW} A_n \alpha_{(i)n} k_{(c)n} (T_{Sn} - T_R) + c_a G_o (T_a - T_R) + HG_{(c)} - HE_s \qquad (6.23)$$

ZN_S : heat capacity of the zone [J/K], T_R : drybulb temperature of the zone [°C], t : time [sec], NW : number of the surfaces in the room
A_n : surface area [m^2], $\alpha_{(i)n}$: film coefficient of the nth surface [W/(m^2 K)], $k_{(c)n}$: convective rate of the film coefficient of a surface [-]
T_{MRT} : mean radiant temperature of the surfaces [°C], c_a : specific heat of the air [J/(kg K)], G_o : ventilation volume [kg/s]
T_a : drybulb temperature of out door air [°C], $HG_{(c)}$: convective heat transfer from indoor heat production element [W]
HE_s : heat supply to the zone [W]

Figure 6.31 shows the constructors of the *Room* class. The list of the zones belonging to the room is an argument.

Table 6.31 The constructors of the Room class

No.	Argument 1	Argument 2
1	List of the zones belonging to the room (Zone array)	Name of the room (string type)
2	List of the zones belonging to the room (Zone array)	-

†1) 詳細な解法については文献 2)を参照

Table 6.32 and 6.33 shows the properties and methods defined in the *Room* class.

There are two type of radiations, the short wave radiation and the long wave radiation. A short wave radiation is a solar radiation transmitted through a window. A long wave radiation is a radiation from wall surface, window surface or other heat production equipment in a room. These two radiations are distributed to wall or window surfaces in a room. The distribution rate can be set by *SetShort*(*Long*)*WaveRadiationRate* method. Initially, distribution rate is set to surface area ratio. The short wave radiation distributed to window surface is conducted back out to outdoor. This amount can be get by *TransmissionHeatLossFromWindow* property.

A radiative heat exchange rate from surface to surface can be set by *SetRadiativeHeatTransferRate* property. These values are calculated by equation 6.24 initially.

$$\Phi_{nj} = \frac{A_j}{\sum_{l}^{NW} A_l} \tag{6.24}$$

Φ_{nj} : radiative heat exchange rate from surface n to surface j [-], A_j : area of the jth surface [m^2]
NW : number of the surfaces belonging to the room [sec]

Table 6.32 The properties defined in the Room class

Name	Description	Has set accessor	Unit	Type
Name	Name	TRUE	-	string
SurfaceNumber	Number of the surfaces	-	-	uint
ZoneNumber	Number of the zones	-	-	uint
TransmissionHeatLossFromWindow	Sum of transmission heat loss from windows in the room	-	W	double
CurrentDateTime	Current date and time	-	-	DateTime

Table 6.33 The methods defined in the Room class

Name	Description			
SetShortWaveRadiationRate	Function	Set a short wave radiation distribution rate [-] of window surface.		
	Return	-		
	Arg.1	Window (Window type)	Arg.2	short wave radiation distribution rate [-]
SetShortWaveRadiationRate	Function	Set a short wave radiation distribution rate [-] of window or wall surface.		
	Return	-		
	Arg.1	Surface (ISurface type)	Arg.2	short wave radiation distribution rate [-]
GetShortWaveRadiationRate	Function	Get a short wave radiation distribution rate [-] of window surface.		
	Return	short wave radiation distribution rate [-]		
	Arg.1	Window (Window type)	Arg.2	-
GetShortWaveRadiationRate	Function	Get a short wave radiation distribution rate [-] of window or wall surface.		
	Return	short wave radiation distribution rate [-]		
	Arg.1	Surface (ISurface type)	Arg.2	-
SetLongWaveRadiationRate	Function	Set a long wave radiation distribution rate [-] of window surface.		
	Return	-		
	Arg.1	Window (Window type)	Arg.2	long wave radiation distribution rate [-]
SetLongWaveRadiationRate	Function	Set a long wave radiation distribution rate [-] of window or wall surface.		
	Return	-		
	Arg.1	Surface (ISurface type)	Arg.2	long wave radiation distribution rate [-]
GetLongWaveRadiationRate	Function	Get a long wave radiation distribution rate [-] of window surface.		
	Return	long wave radiation distribution rate [-]		
	Arg.1	Window (Window type)	Arg.2	-
GetLongWaveRadiationRate	Function	Get a long wave radiation distribution rate [-] of window or wall surface.		
	Return	long wave radiation distribution rate [-]		
	Arg.1	Surface (ISurface type)	Arg.2	-
SetRadiativeHeatTransferRate	Function	Set a radiative heat exchange rate [-] from surface 1 to surface 2		
	Return	-		
	Arg.1	Surface 1 (ISurface type)	Arg.2	Surface 2 (ISurface type)
	Arg.3	radiative heat exchange rate [-] from surface 1 to surface 2	-	-
GetRadiativeHeatTransferRate	Function	Get a radiative heat exchange rate [-] from surface 1 to surface 2		
	Return	radiative heat exchange rate [-] from surface 1 to surface 2		
	Arg.1	Surface 1 (ISurface type)	Arg.2	Surface 2 (ISurface type)

The constructor of the *MultiRoom* class takes a list of *Room* objects as an argument.

```
MultiRoom mRoom = new MultiRoom(rooms);
```

Table 6.34 shows the methods defined in the *MultiRoom* class. Interior ventilation volume can be set by *SetAirFlow* method. Use *UpdateRoomTemperatures* method and *UpdateRoomHumidities* method to update room state.

Table 6.34 The methods defined in the MultiRoom class

Name	Description			
UpdateRoomTemperatures	Function	Update temperatures of the rooms.		
	Return	-		
	Arg.1	-	Arg.2	-
UpdateRoomHumidities	Function	Update humidity ratio of the rooms.		
	Return	-		
	Arg.1	-	Arg.2	-
SetTimeStep	Function	Set the time step [sec]		
	Return	-		
	Arg.1	Time step [sec]	Arg.2	-
SetAirFlow	Function	Set the air flow [m^3/h] between the rooms		
	Return	-		
	Arg.1	Upstream zone (ImmutableZone type)	Arg.2	Downstream zone (ImmutableZone type)
	Arg.3	Air flow [m^3/h]	Arg.4	-
GetAirFlow	Function	Get the air flow [m^3/h] between the rooms		
	Return	Air flow [m^3/h]		
	Arg.1	Upstream zone (ImmutableZone type)	Arg.2	Downstream zone (ImmutableZone type)
SetCurrentDateTime	Function	Set current date and time.		
	Return	-		
	Arg.1	Current date and time (DateTime type)	Arg.2	-

Figure 6.30 shows a sample program which calculates an air state and a heat load of the building discussed at 1) with *MultiRoom* class. Most of the program codes are same as the codes shown in figure 6.28.

The instances of Room class and MultiRoom class is created in the line 187~191. The west room is consists of west perimeter zone and west interior zone. The east room is consists of east perimeter zone and east interior zone.

Ventilation volumes are set in the line 193~197.

A short wave radiation distribution rates are set in the line 199~209. It assumes that 60% of short wave radiation is distributed to perimeter floor. Distribution rates to other surfaces are set to surface area ratio.

24 hours calculation is iterated 100 times to get steady state result.

```
/// <summary>Sample program calculating the air state and heat load of the building (MultiRoom class)</summary>
private static void AirStateAndHeatLoadTest2()
{
   //A sample weather data
   //Drybulb temperature [C]
   double[] dbt = new double[] { 24.2, 24.1, 24.1, 24.2, 24.3, 24.2, 24.4, 25.1, 26.1, 27.1, 28.8, 29.9,
      30.7, 31.2, 31.6, 31.4, 31.3, 30.8, 29.4, 28.1, 27.5, 27.1, 26.6, 26.3 };
   //Humidity ratio [kg/kg(DA)]
   double[] hum = new double[] { 0.0134, 0.0136, 0.0134, 0.0133, 0.0131, 0.0134, 0.0138, 0.0142, 0.0142, 0.0140, 0.0147, 0.0149,
      0.0142, 0.0146, 0.0140, 0.0145, 0.0144, 0.0146, 0.0142, 0.0136, 0.0136, 0.0135, 0.0136, 0.0140 };
   //Nocturnal radiation [W/m2]
   double[] nrd = new double[] { 32, 30, 30, 29, 26, 24, 24, 25, 25, 25, 24, 24, 24, 23, 24, 24, 24, 24, 23, 23, 24, 26, 25, 23 };
   //Direct normal radiation [W/m2]
   double[] dnr = new double[] { 0, 0, 0, 0, 0, 0, 106, 185, 202, 369, 427, 499, 557, 522, 517, 480, 398, 255, 142, 2, 0, 0, 0, 0 };
   //Diffuse horizontal radiation [W/m2]
   double[] drd = new double[] { 0, 0, 0, 0, 0, 0, 36, 115, 198, 259, 314, 340, 340, 349, 319, 277, 228, 167, 87, 16, 0, 0, 0, 0 };

   //Create an instance of the Outdoor class
   Outdoor outdoor = new Outdoor();
   Sun sun = new Sun(Sun.City.Tokyo);  //Located in Tokyo
   outdoor.Sun = sun;
   outdoor.GroundTemperature = 25;     //Ground temperature is assumed to be constant

   //Create an instance of the Incline class
   Incline nIn = new Incline(Incline.Orientation.N, 0.5 * Math.PI); //North, Vertical
   Incline eIn = new Incline(Incline.Orientation.E, 0.5 * Math.PI); //East, Vertical
   Incline wIn = new Incline(Incline.Orientation.W, 0.5 * Math.PI); //West, Vertical
   Incline sIn = new Incline(Incline.Orientation.S, 0.5 * Math.PI); //South, Vertical
   Incline hIn = new Incline(Incline.Orientation.S, 0);  //Horizontal

   //Create an instance of the Zone class
   Zone[] zones = new Zone[4];
   Zone wpZone = zones[0] = new Zone("West perimeter zone");
   wpZone.Volume = 3 * 5 * 3;  //Ceiling height is 3m
   Zone wiZone = zones[1] = new Zone("West interior zone");
   wiZone.Volume = 4 * 5 * 3;
   Zone epZone = zones[2] = new Zone("East perimeter zone");
   epZone.Volume = 3 * 5 * 3;
   Zone eiZone = zones[3] = new Zone("East interior zone");
   eiZone.Volume = 4 * 5 * 3;
   foreach (Zone zn in zones)
   {
      zn.VentilationVolume = 10; //Ventilation volume[CMH]
      zn.TimeStep = 3600;
      zn.DrybulbTemperatureSetPoint = 26;
      zn.HumidityRatioSetPoint = 0.01;
   }

   //Set a heat production element to the east interior zone
   //Convective sensible heat=100W, Radiative sensible heat=100W, Latent heat=20W
   eiZone.AddHeatGain(new ConstantHeatGain(100, 100, 20));

   //Create an instance of the WallLayers class : Concrete,400mm
   WallLayers wl = new WallLayers();
   wl.AddLayer(new WallLayers.Layer(new WallMaterial(WallMaterial.PredefinedMaterials.ReinforcedConcrete), 0.4));

   //Create an instance of the GlassPanes class:Low-emissivity coating single glass
   GlassPanes gPanes = new GlassPanes(new GlassPanes.Pane(GlassPanes.Pane.PredifinedGlassPane.HeatReflectingGlass06mm));

   //Set wall surfaces to the zone objects
   Wall[] walls = new Wall[18];
   List<WallSurface> outdoorSurfaces = new List<WallSurface>();
   Wall wpwWall = walls[0] = new Wall(wl, "West wall in the west perimeter zone");
   wpwWall.SurfaceArea = 3 * 3;
   outdoorSurfaces.Add(wpwWall.GetSurface(true));
   wpZone.AddSurface(wpwWall.GetSurface(false));
   wpwWall.SetIncline(wIn, true);

   Wall wpcWall = walls[1] = new Wall(wl, "Ceiling in the west perimeter zone");
   wpcWall.SurfaceArea = 3 * 5;
   outdoorSurfaces.Add(wpcWall.GetSurface(true));
   wpZone.AddSurface(wpcWall.GetSurface(false));
   wpcWall.SetIncline(hIn, true);

   Wall wpfWall = walls[2] = new Wall(wl, "Floor in the west perimeter zone");
   wpfWall.SurfaceArea = 3 * 5;
   outdoor.AddGroundWallSurface(wpfWall.GetSurface(true));
   wpZone.AddSurface(wpfWall.GetSurface(false));

   Wall winWall = walls[3] = new Wall(wl, "North wall in the west interior zone");
```

```
    winWall.SurfaceArea = 3 * 5;
    outdoorSurfaces.Add(winWall.GetSurface(true));
    wiZone.AddSurface(winWall.GetSurface(false));
    winWall.SetIncline(nIn, true);

    Wall wiwWall = walls[4] = new Wall(wl, "West wall in the west interior zone");
    wiwWall.SurfaceArea = 3 * 4;
    outdoorSurfaces.Add(wiwWall.GetSurface(true));
    wiZone.AddSurface(wiwWall.GetSurface(false));
    wiwWall.SetIncline(wIn, true);

    Wall wicWall = walls[5] = new Wall(wl, "Ceiling in the west interior zone");
    wicWall.SurfaceArea = 4 * 5;
    outdoorSurfaces.Add(wicWall.GetSurface(true));
    wiZone.AddSurface(wicWall.GetSurface(false));
    wicWall.SetIncline(hIn, true);

    Wall wifWall = walls[6] = new Wall(wl, "Floor in the west interior zone");
    wifWall.SurfaceArea = 4 * 5;
    outdoor.AddGroundWallSurface(wifWall.GetSurface(true));
    wiZone.AddSurface(wifWall.GetSurface(false));

    Wall epwWall = walls[7] = new Wall(wl, "East wall in the east perimeter zone");
    epwWall.SurfaceArea = 3 * 3;
    outdoorSurfaces.Add(epwWall.GetSurface(true));
    epZone.AddSurface(epwWall.GetSurface(false));
    epwWall.SetIncline(eIn, true);

    Wall epcWall = walls[8] = new Wall(wl, "Ceiling in the east perimeter zone");
    epcWall.SurfaceArea = 3 * 5;
    outdoorSurfaces.Add(epcWall.GetSurface(true));
    epZone.AddSurface(epcWall.GetSurface(false));
    epcWall.SetIncline(hIn, true);

    Wall epfWall = walls[9] = new Wall(wl, "Floor in the east perimeter zone");
    epfWall.SurfaceArea = 3 * 5;
    outdoor.AddGroundWallSurface(epfWall.GetSurface(true));
    epZone.AddSurface(epfWall.GetSurface(false));

    Wall einWall = walls[10] = new Wall(wl, "North wall in the east interior zone");
    einWall.SurfaceArea = 5 * 3;
    outdoorSurfaces.Add(einWall.GetSurface(true));
    eiZone.AddSurface(einWall.GetSurface(false));
    einWall.SetIncline(nIn, true);

    Wall eiwWall = walls[11] = new Wall(wl, "East wall in the east interior zone");
    eiwWall.SurfaceArea = 4 * 3;
    outdoorSurfaces.Add(eiwWall.GetSurface(true));
    eiZone.AddSurface(eiwWall.GetSurface(false));
    eiwWall.SetIncline(eIn, true);

    Wall eicWall = walls[12] = new Wall(wl, "Ceiling in the east interior zone");
    eicWall.SurfaceArea = 4 * 5;
    outdoorSurfaces.Add(eicWall.GetSurface(true));
    eiZone.AddSurface(eicWall.GetSurface(false));
    eicWall.SetIncline(hIn, true);

    Wall eifWall = walls[13] = new Wall(wl, "Floor in the east interior zone");
    eifWall.SurfaceArea = 4 * 5;
    outdoor.AddGroundWallSurface(eifWall.GetSurface(true));
    eiZone.AddSurface(eifWall.GetSurface(false));

    Wall cpWall = walls[14] = new Wall(wl, "Inner wall at perimeter");
    cpWall.SurfaceArea = 3 * 3;
    wpZone.AddSurface(cpWall.GetSurface(true));
    epZone.AddSurface(cpWall.GetSurface(false));

    Wall ciWall = walls[15] = new Wall(wl, "Inner wall at interior");
    ciWall.SurfaceArea = 4 * 3;
    wiZone.AddSurface(ciWall.GetSurface(true));
    eiZone.AddSurface(ciWall.GetSurface(false));

    Wall wpsWall = walls[16] = new Wall(wl, "South wall in the west perimeter zone");
    wpsWall.SurfaceArea = 5 * 3 - 3 * 2;    //Reduce window surface area
    outdoorSurfaces.Add(wpsWall.GetSurface(true));
    wpZone.AddSurface(wpsWall.GetSurface(false));
    wpsWall.SetIncline(sIn, true);

    Wall epsWall = walls[17] = new Wall(wl, "South wall in the east perimeter zone");
    epsWall.SurfaceArea = 5 * 3 - 3 * 2;    //Reduce window surface area
    outdoorSurfaces.Add(epsWall.GetSurface(true));
```

```
    epZone.AddSurface(epsWall.GetSurface(false));
    epsWall.SetIncline(sIn, true);

    //Initialize outdoor surfaces
    foreach (WallSurface ws in outdoorSurfaces)
    {
        //Add wall surfaces to Outdoor object
        outdoor.AddWallSurface(ws);
        //Initialize emissivity of surface
        ws.InitializeEmissivity(WallSurface.SurfaceMaterial.Concrete);
    }

    //Add windows to the west zone
    Window wWind = new Window(gPanes, "Window in the west perimeter zone");
    wWind.SurfaceArea = 3 * 2;
    wpZone.AddWindow(wWind);
    outdoor.AddWindow(wWind);
    //Add windows to the east zone
    Window eWind = new Window(gPanes, "Window in the east perimeter zone");
    eWind.SurfaceArea = 3 * 2;
    //Set horizontal sun shade.
    eWind.Shade = SunShade.MakeHorizontalSunShade(3, 2, 1, 1, 1, 0.5, sIn);
    wpZone.AddWindow(eWind);
    outdoor.AddWindow(eWind);

    //Creat an insances of the Room class and MultiRoom class
    Room eRm = new Room(new Zone[] { epZone, eiZone }); //East room
    Room wRm = new Room(new Zone[] { wpZone, wiZone }); //Weast room
    MultiRoom mRoom = new MultiRoom(new Room[] { eRm, wRm }); //Multi room (east and west rooms)
    mRoom.SetTimeStep(3600);

    //Set ventilation volume
    wpZone.VentilationVolume = 10; //Only west perimeter zone has outdoor air ventilation
    mRoom.SetAirFlow(wpZone, wiZone, 10);
    mRoom.SetAirFlow(epZone, eiZone, 10);
    mRoom.SetAirFlow(eiZone, epZone, 10);

    //Set short wave radiation distribution:60% of short wave is distributed to perimeter floor.
    double sfSum = 0;
    foreach (ISurface isf in eRm.GetSurface()) sfSum += isf.Area;
    sfSum -= epfWall.SurfaceArea;
    foreach (ISurface isf in eRm.GetSurface()) eRm.SetShortWaveRadiationRate(isf, isf.Area / sfSum * 0.4);
    eRm.SetShortWaveRadiationRate(epfWall.GetSurface(false), 0.6);
    sfSum = 0;
    foreach (ISurface isf in wRm.GetSurface()) sfSum += isf.Area;
    sfSum -= wpfWall.SurfaceArea;
    foreach (ISurface isf in wRm.GetSurface()) wRm.SetShortWaveRadiationRate(isf, isf.Area / sfSum * 0.4);
    wRm.SetShortWaveRadiationRate(wpfWall.GetSurface(false), 0.6);

    //Output title wrine to standard output stream
    StreamWriter sWriter = new StreamWriter("AirStateAndHeatLoadTest2.csv");
    foreach (Zone zn in zones) sWriter.Write(zn.Name + "Drybulb temperature[C], " + zn.Name +
                        "Humidity ratio[kg/kgDA], " + zn.Name + "Sensible heat load[W], " + zn.Name + "Latent heat load[W], ");
    sWriter.WriteLine();

    //Update the state (Iterate 100 times to make state steady)
    for (int i = 0; i < 100; i++)
    {
        DateTime dTime = new DateTime(2007, 8, 3, 0, 0, 0);
        for (int j = 0; j < 24; j++)
        {
            //Set date and time to Sun and Zone object.
            sun.Update(dTime);
            mRoom.SetCurrentDateTime(dTime);

            //Operate HVAC system (8:00~19:00)
            bool operating = (8 <= dTime.Hour && dTime.Hour <= 19);
            foreach (Zone zn in zones)
            {
                zn.ControlHumidityRatio = operating;
                zn.ControlDrybulbTemperature = operating;
            }

            //Set weather state.
            outdoor.AirState = new MoistAir(dbt[j], hum[j]);
            outdoor.NocturnalRadiation = nrd[j];
            sun.SetGlobalHorizontalRadiation(drd[j], dnr[j]);

            //Set ventilation air state.
            wpZone.VentilationAirState = outdoor.AirState;

```

```
243         //Update boundary state of outdoor facing surfaces.
244         outdoor.SetWallSurfaceBoundaryState();
245
246         //Update the walls.
247         foreach (Wall wal in walls) wal.Update();
248
249         //Update the MultiRoom object.
250         mRoom.UpdateRoomTemperatures();
251         mRoom.UpdateRoomHumidities();
252
253         //Update date and time
254         dTime = dTime.AddHours(1);
255
256         //If it is last iteration, output result to CSV text.
257         if (i == 99)
258         {
259             foreach (Zone zn in zones)
260             {
261                 sWriter.Write(zn.CurrentDrybulbTemperature.ToString("F1") + ", " + zn.CurrentHumidityRatio.ToString("F3") + ", " +
262                 zn.CurrentSensibleHeatLoad.ToString("F0") + ", " + zn.CurrentLatentHeatLoad.ToString("F0") + ", ");
263             }
264             sWriter.WriteLine();
265         }
266     }
267   }
268
269   sWriter.Close();
270 }
```

Figure 6.30 The sample program calculating the air state and the heat load with MultiRoom class

Figure 6.31 and figure 6.32 shows the result of the simulation. The heat load of the west interior zone is affected by ventilation at the perimeter zone, since the ventilation between the interior zone and the perimeter zone is explicitly solved in the *MultiRoom* class.

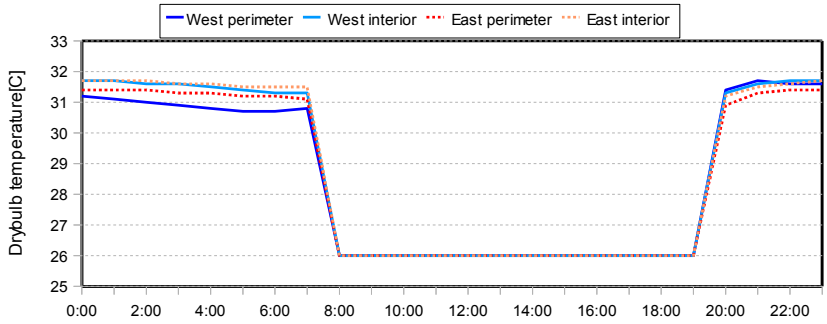


Figure 6.31 乾球温度の推移

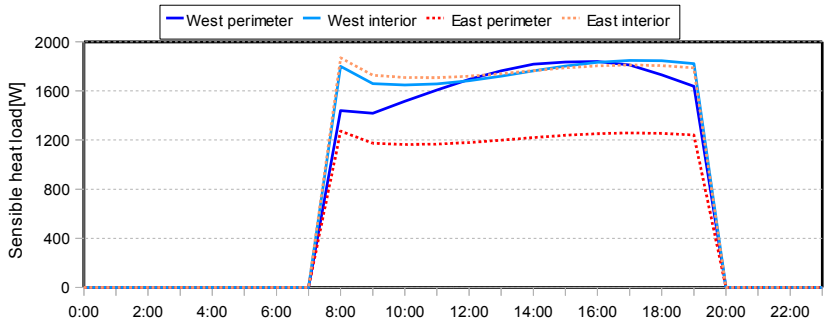


Figure 6.32 顕熱負荷の推移

1) Daniel R. Clark : HVACSIM building systems and equipment simulation program reference manual
2) 宇田川光弘 : パソコンによる空気調和計算法, オーム社, 1986
3) Hiroyasu Okuyama : Bouilding Heat and Air Transfer Models Based on System Theory, Bulletin of the Japan Society for Industrial and Applied Mathematics, 13(1), pp.61-71, 2003
4) A.P.Gagge et al. : A standard predictive index of human response to the thermal environment, AHRAE Transaction, Vol.93, 1987
5) S.Tanabe, K.Kobayashi, J.Nakano, Y.Ozeki et al.. 2002. Evaluation of thermal comfort using combined multi-node thermoregulation (65MN) and radiation models and computational fluid dynamics (CFD). Energy and Buildings. No.34. pp.637-646.
6) Kimura, K., Stephenson, D.G. : Solar radiation on cloudy days, ASHRAE Transactions, Vol. 75, Part 1 (1969), pp.227~234
7) Bugler, J.W. : The determination of hourly insolation on an inclined plane using a diffuse irradiance model based on hourly measured global horizontal insolation, Solar Energy, 19 (1977), pp.477~491
8) Orgill, J.F., Hollands, K.G.T. : Correlation equation for hourly diffuse radiation on a horizontal surface, Solar Energy, 19-4 (1977), pp.357~359
9) Spencer, J.W. : A comparison of methods for estimating hourly diffuse solar radiation from global solar radiation, Solar Energy, 29-1 (1982), pp.19~32
10) Berlage,Von H.P.:Zur Theorie der Beleuchtung einer horizontalen Flache durch Tageslicht,Meteorologische Zeitschrift, May 1928,pp.174-180
11) 松尾陽:日本建築学会論文報告集,快晴時の日射について 日射量に関する研究 2,pp.21-24,1960
12) 永田忠彦:晴天空による水平面散乱の日射の式の試案,日本建築学会学術講演梗概集,1978
13) Liu,B.Y.H & Jordan,R.C:The interrelationship and characteristic distribution of direct, diffuse and total solar radiation, solar energy, Vol.4, No.3, 1960
14) 宇田川光弘,木村建一:水平面全天日射量観測値よりの直達日射量の推定,日本建築学会論文報告集,No.267,pp.83-90,1978, 0530
15) 渡辺俊行:水平面全天日射量の直散分離と傾斜面日射量の推定,日本建築学会論文報告集,No.330,pp.96-108,19830830
16) H.Akasaka:Model of circumsolar radiation and diffuse sky radiation including cloudy sky, ISES, Solar World Congress, 1991
17) 三木信博:標準気象データの日射直散分離に関する研究 その 6 日射直散分離法の提案,日本建築学会学術講演梗概集,pp.857-858,1991
18) 松尾陽 : 空調負荷計算におけるふく射伝熱の扱い, 空気調和・衛生工学, 59-4, 1985, pp.323~329
19) 木村建一, 伊藤直明 : 事務所建築の家具の熱容量, 日本建築学会関東支部第 29 会研究発表会, 1961-1, 同続報, 同第 34 回, 1963-5
20) 石野久彌, 郡公子 : 事務所建築における家具類の熱的影響に関する実測・実験研究, 日本建築学会計画系論文報告集 (372), 59-66, 1987, 0228