```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;

/*
 * Title:   Calculator
 * Author:  Paul McKillop
 * Date:    November 2020
 * Purpose: Provide calculation methods
 */

namespace FlooringCalculator.Models
{
    public class Calculator
    {
        /// <summary>
        /// Number of tiles to cover whole floor regardless of cutouts;
        /// number is rounded up to guarantee full cover.
        /// Wherever division is used, it's wise to check for divide by zero.
        /// </summary>
        /// <param name="room"></param>
        /// <param name="tile"></param>
        /// <returns>int of number of tiles</returns>
        public int NumberTilesWholeFloor(Room room, Tile tile)
        {
            int tempTotal = 0;

            try
            {
                var wideQuantity = room.RoomWide / tile.TileWide;
                int wideQuantityUp = Convert.ToInt32(Math.Ceiling(wideQuantity));

                var longQuantity = room.RoomLong / tile.TileLong;
                int longQuantityUp = Convert.ToInt32(Math.Ceiling(longQuantity));

                tempTotal = wideQuantityUp * longQuantityUp;
            }
            catch (DivideByZeroException ex)
            {
                MessageBox.Show("Attempted divide by Zero");
                Console.WriteLine("Attempted divide by zero. {0}", ex.Message);
            }

            return tempTotal;
        }

        /// <summary>
        /// Number of tiles save by the cutout areas;
        /// number of tiles is rounded down to avoid overestimation
        /// </summary>
        /// <param name="room"></param>
        /// <param name="tile"></param>
        /// <returns>int number of tiles</returns>
        public int NumberTilesCutouts(Room room, Tile tile)
        {
            int tempTotal = 0;

            try
            {
                // -- Cutout 1
                var cutout1WideQuantity = room.Cutout1Wide / tile.TileWide;
                int cutout1WideQuantityDown = Convert.ToInt32(Math.Floor(
                cutout1WideQuantity));

                var cutout1LongQuantity = room.Cutout1Long / tile.TileLong;
                int cutout1LongQuantityDown = Convert.ToInt32(Math.Floor(
                cutout1LongQuantity));

                int tilesCutout1 = cutout1LongQuantityDown * cutout1WideQuantityDown;
```

```
            // -- Cutout 2
            var cutout2WideQuantity = room.Cutout2Wide / tile.TileWide;
            int cutout2WideQuantityDown = Convert.ToInt32(Math.Floor(
            cutout2WideQuantity));

            var cutout2LongQuantity = room.Cutout2Long / tile.TileLong;
            int cutout2LongQuantityDown = Convert.ToInt32(Math.Floor(
            cutout2LongQuantity));

            int tilesCutout2 = cutout2LongQuantityDown * cutout2WideQuantityDown;

            // -- Calculate total
            tempTotal = tilesCutout1 + tilesCutout2;
        }
        catch (DivideByZeroException ex)
        {
            MessageBox.Show("Attempted divide by Zero");
            Console.WriteLine("Attempted divide by zero. {0}", ex.Message);
        }

        return tempTotal;
    }


    /// <summary>
    /// Calculates the number of tile needed
    /// </summary>
    /// <param name="room"></param>
    /// <param name="tile"></param>
    /// <returns>int for number of tiles</returns>
    public int NumberTilesForFloor(Room room, Tile tile)
    {
        int tilesWholeFloor = NumberTilesWholeFloor(room, tile);
        int tilesCutoutAreas = NumberTilesCutouts(room, tile);

        // -- two way of doing something
        //int tilesNeededForFloor = NumberTilesWholeFloor(room, tile) -
        NumberTilesCutouts(room, tile);
        int tilesNeededForFloor = tilesWholeFloor - tilesCutoutAreas;


        return tilesNeededForFloor;
    }

    /// <summary>
    /// Calculates the area leftover from cuts when tiles used
    /// to cover all floor
    /// </summary>
    /// <param name="room"></param>
    /// <param name="tile"></param>
    /// <returns>decimal of area</returns>
    public decimal AreaLeftoverTile(Room room, Tile tile)
    {
        decimal tempTotal = 0;

        try
        {
            var wholeRoomArea = RoomAreas.WholeRoomArea(room);
            var oneTileArea = tile.TileLong * tile.TileWide;
            var areaTilesUsed = oneTileArea * NumberTilesWholeFloor(room, tile);

            tempTotal = areaTilesUsed - wholeRoomArea;
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
            throw;
        }
```

```
                return tempTotal;

            }
        }
    }
```