```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using FirstApp.Models;

namespace FirstApp.BLL
{
    public class Setup
    {
        static private List<Tooling> wholeSetup = new List<Tooling>();
        static private List<Tooling> fractionSetup = new List<Tooling>();

        public List<Tooling> CalcSetup(decimal target, string mach)
        {
            List<Tooling> setup = new List<Tooling>();

            wholeSetup.Clear();
            fractionSetup.Clear();

            // Get tooling inventory based on machine
            List<decimal> spacers = Invtry.GetArborSpacers(mach);

            // Break target into whole and fraction
            var parts = target.ToString().Split('.');

            decimal whole = decimal.Parse(parts[0]);
            decimal fraction = decimal.Parse(parts[1])/1000;

            // If fraction tenths = 0,1 or 2, borrow from whole
            if ((whole >= 1) && (fraction * 10 <= 2))
            {
                whole -= 1;
                fraction += 1;
            }

            if (whole >= 1)
                wholeSetup = Sum_Whole(spacers, whole);

            fractionSetup = Sum_Fraction_Recursive(spacers, fraction, new
              List<decimal>());

            //Add fractionSetup to end of wholeSetup.  Preserve order.
            //wholeSetup.AddRange(fractionSetup);

            setup.AddRange(wholeSetup);
            setup.AddRange(fractionSetup);

            // return setup;
            return setup;
        }

        //pass in list of spacers and whole part of mult width
        public static List<Tooling> Sum_Whole(List<decimal> spacers, decimal mult)
        {
            decimal startMult = mult;

            //List<Tooling> wholeSetup = new List<Tooling>();

            for (int i = 0; i < spacers.Count; i++)
            {
                Tooling t = new Tooling();

                decimal n = spacers[i];

                // Find largest integer <= mult / current spacer
                decimal numSp = Math.Floor(mult / n);

                // if spacer > mult, numSp = 0
                // don't execute until spacer < mult
                if (numSp != 0)
                {
                    // Create tooling object
                    t.loc = "arbor";
                    t.tp = "whole";
                    t.qty = Convert.ToInt16(numSp);
                    t.sz = n;

                    // Add tooling object to list
                    wholeSetup.Add(t);

                    // subract spacer from mult balance
                    mult = mult - numSp * n;
                }
            }

            return wholeSetup;
        }

        // pass in spacers, fraction part of mult width and empty list for recursion
        // not sure what is happening on the recursion
        public static List<Tooling> Sum_Fraction_Recursive(List<decimal> spacers, decimal mult, List<decimal> partial)
        {
            decimal s = 0;
            foreach (decimal x in partial)
                s += x;

            if (s == mult)
            {
                Console.WriteLine("Spacers(" + string.Join(",", partial.ToArray
                  ()) + ") = " + mult);

                if (fractionSetup.Count == 0)
                {
                    for (int k = 0; k < partial.Count; k++)
                    {
                        Tooling t = new Tooling();

                        decimal sp = partial[k];

                        // Create tooling object
                        t.loc = "arbor";
                        t.tp = "frac";
                        t.qty = 1;
                        t.sz = sp;

                        fractionSetup.Add(t);
                    }
                }
            }

            if (s >= mult)
                return fractionSetup;


            for (int i = 0; i < spacers.Count; i++)
                {
                    List<decimal> remaining = new List<decimal>();

                    decimal n = spacers[i];

                    for (int j = i + 1; j < spacers.Count; j++)
                        remaining.Add(spacers[j]);

                    List<decimal> partial_rec = new List<decimal>(partial);
                    partial_rec.Add(n);

                    Sum_Fraction_Recursive(remaining, mult, partial_rec);
                }


            return fractionSetup;
        }
    }
}

```