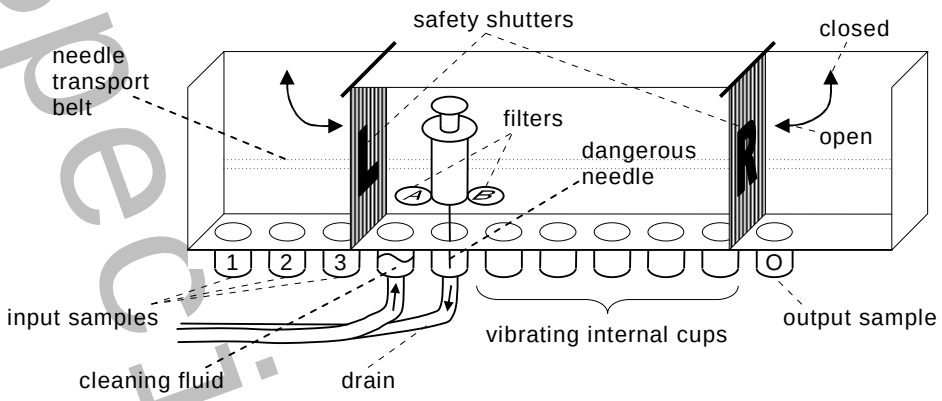
Mixin

Mixin is world leader amongst producers of mixing machines for the medical sector. The key selling points of their machines are high-safety, operation-efficient, low-contamination mixing and their Tailored Mixing Process® concept.

The high safety of Mixin® mixing machines is obtained by two safety shutters that always protect the operator from the dangerous needle that is used to perform the mixing. Operation efficiency is obtained by transporting the input samples (with a maximum of 3 samples) as quickly as possible into the machine so that the operator can start providing new input samples as early as possible. A similar thing goes for the output sample. Low contamination is obtained by cleaning the needle each time it extracts a sample with a different composition or if it extracts through a different filter, combined with a nano-coating in the cups that guarantees that no fluid remains if a cup is emptied.



All Mixin® machines are based on a single, fixed physical layout. The differences between the machines that are delivered to customers are in the software that controls the machines. In Mixin®'s software the following class is used as basis for the construction of mixing machines for customers:

```
class MixingMachine
{   MixingMachine()                {…} // Initialises the mixing machine:
                                       // empties all internal cups, cleans the needle,
                                       // places it above the drain, removes any filter
                                       // and opens both shutters

    void   move(int offset)        {…} // Moves needle 'offset' positions (- = L, + = R)
    void   filt(int filter)        {…} // Applies 'filter' (0 = no filter, 1 = A, 2 = B)
    double scan()                  {…} // Scans amount (mls) in the cup beneath the needle
    void   suck(double amount)     {…} // Extracts 'amount' mls from the cup
    void   blow(double amount)     {…} // Injects 'amount' mls into the cup
    void   open(int shutter)       {…} // Opens shutter 'shutter' (0 = L, 1 = R)
    void   shut(int shutter)       {…} // Closes shutter 'shutter' (0 = L, 1 = R)
    void   wait(int time)          {…} // Waits 'time' seconds (+/- 2 seconds)
}
```

To implement a mixing machine for a particular customer, the above class is extended. For each mixing process a method is added that uses the methods of the MixingMachine class to control the machine.

```
class DNAMixingMachine extends MixingMachine
{   int dna_cosfultation()    {…}
    int dna_retrubfacsation() {…}
    int dna_trimidopasation() {…}
}
```

The number of developers that Mixin® needs in order to fulfill the promises of the Tailored Mixing Process® is vastly increasing. Also many needles (€400.000,- each) have been broken due to bugs in implemented mixing processes. There is one known case where an operator has died from being hit by a needle. An unknown number of people have died from errors in mixing processes. These problems have made the board of directors decide to switch to model-driven software development (MDSD) because models are much easier to design and analyse for deficiencies than code.

Mixin® is looking for the best party to implement MDSD in their software development process. They ask each party to develop a domain-specific language (DSL) for their mixing process. As input they provide the Java code for three different DNA mixing processes, implemented by their best software developers. They ask each party to show what these three pieces of code would look like in their DSL. The party may use any tools to draw the three models (hand-drawn models are also accepted) as long as the files that are sent are in pdf format. A small explanation of the language should also be provided. The files should be sent before June 18th, 2008 to info@ictnoviq.nl with subject "Mixin", stating your name(s), company, e-mail address(es) and (optional) phone number(s). The designer(s) of the best DSL will be rewarded with a nice present and, of course, eternal fame.

```
class DNAMixingMachine extends MixingMachine
{   int dna_cosfultation()
    {   double i,j;
        shut(0);
        move(-4);
        if(scan()!=9) return -1;
        suck(9);move(5);blow(9);
        move(-2);suck(30);move(1);blow(30);
        move(-3);
        if(scan()<6) return -1;
        i=scan();suck(i);move(5);open(0);blow(i);
        move(-3);suck(30);move(1);blow(30);
        move(1);filt(1);suck(3);move(3);filt(0);blow(3);
        move(-5);suck(30);move(1);blow(30);
        move(2);filt(2);suck(2);
        move(2);filt(0);blow(2);
        wait(8);
        move(-5);suck(30);move(1);blow(30);
        move(1);filt(1);suck(3);move(3);filt(0);blow(3);
        move(-5);suck(30);move(1);blow(30);
        move(2);filt(2);suck(2);
        move(2);filt(0);blow(2);
        wait(8);
        move(-5);suck(30);move(1);blow(30);
        move(1);filt(1);suck(3);move(3);filt(0);blow(3);
        move(-5);suck(30);move(1);blow(30);
        move(2);filt(2);suck(2);
        move(2);filt(0);blow(2);
        wait(8);
        move(-5);suck(30);move(1);blow(30);
        move(2);i=scan();suck(i);move(3);
        if(i<=10) {blow(i);j=i;}
        else {blow(10);j=10;}
        move(-1);blow(i-j);
        wait(7);
        move(-5);suck(30);move(1);blow(30);
        move(4);suck(3+2+3+2+3+2+i-j);move(1);blow(3+2+3+2+3+2+i-j);
        wait(12);
        suck(3+2+3+2+3+2+i);shut(1);move(1);blow(3+2+3+2+3+2+i);
        move(-7);open(1);suck(30);move(1);blow(30);
        return 0;
    }
```

```
int dna_retrubfacsation()
{   double j;
    double h=0;
    shut(0);
    for(int i = 0; i < 3; i = i+1)
    {   move(i-4);
        j=scan();h=h+j;if(h<=30){suck(j);}else{return -1;}move(5);
        if(i==2) open(0);
        h=h-j;if(h>=0){blow(j);}else{return -1;}
        move(-2-i);
        if(h==0){suck(30);move(1);blow(30);}else{return -1;}
    }
    move(2);j=scan();filt(1);
    h=h+(.50*j);if(h<=30){suck(.50*j);}else{return -1;}
    move(-1);filt(0);
    double l=scan();h=h-(.25*j);if(h>=0){blow(.25*j);}else{return -1;}move(2);
    double k=scan();h=h-(.25*j);if(h>=0){blow(.25*j);}else{return -1;}
    move(-4);if(h==0){suck(30);move(1);blow(30);}else{return -1;}
    move(2);filt(2);h=h+(.5*j);if(h<=30){suck(.5*j);}else{return -1;}
    move(1);filt(0);h=h-(.5*j);if(h>=0){blow(.5*j);}else{return -1;}
    wait(12);
    h=h+(.6*j+.8*k);if(h<=30){suck(.6*j+.8*k);}else{return -1;}
    move(-2);h=h-(.6*j+.8*k);if(h>=0){blow(.6*j+.8*k);}else{return -1;}
    wait(17);
    move(2);suck(.2*k+.15*j);move(-3);blow(.2*k+.15*j);
    move(-1);suck(30);move(1);blow(30);
    move(1);suck(l+.85*j+.8*k);shut(1);move(5);blow(l+.85*j+.8*k);move(-7);open(1);
    suck(30);move(1);blow(30);
    return 0;
}
```

```
    int dna_trimidopasation()
    {   int result = 0;
        double j;
        shut(0);
        for(int i = 0; i < 3; i = i+1)
        {   if (result == 0)
            {   move(i-4);
                j = scan();
                if (j < 4) result = -1;
                else
                {   suck(j);
                    move(5);
                    if(i==2) open(0);
                    blow(4);
                    move(-1-i);
                    blow(j-4);
                    move(-1);
                    suck(30);
                    move(1);
                    blow(30);
                }
            }
        }
        if (result == 0)
        {   for (int k = 0; k < 4 ; k = k+1)
            {   for (int l = 0; l < 3 ; l = l+1)
                {   move(l+1);
                    suck(1);
                    move(3-l);
                    blow(1);
                    move(-5);
                    suck(30);
                    move(1);
                    blow(30);
                }
                wait(3);
            }
            move(4);
            suck(12);
            shut(1);
            move(2);
            blow(12);
            move(-7);
            open(1);
            suck(30);
            move(1);
            blow(30);
        }
        return result;
    }
}
```