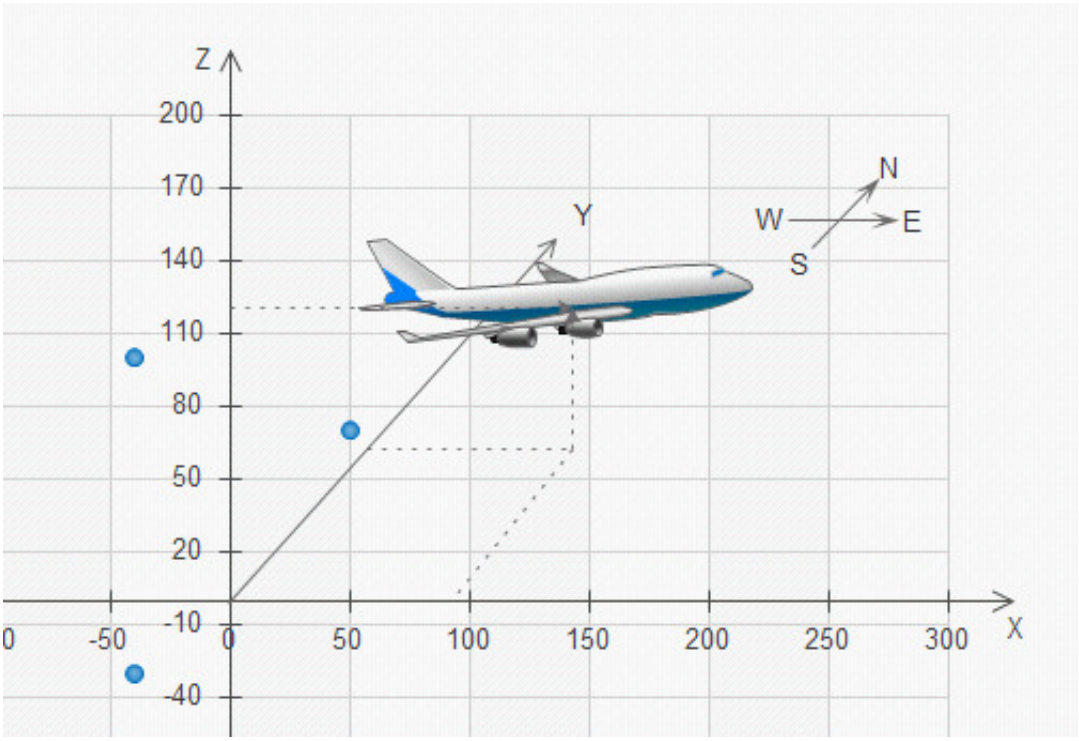
Unmanned-aircraft

- 无人机(Unmanned Aircraft)，又称无人驾驶机，是利用无线电遥感和自备程序控制的不载人飞机。
- Z公司是一家无人机制造商，拟制造一款简易的无人机，它可以接受无线遥感指令，完成简单的动作。
- 工程师使用四元组(x,y,z,d)表示无人机位置，其中(x,y,z)为其三维坐标位置，d为其朝向(包括**E**ast,**S**outh,**W**est,**N**orth)。
- Aircraft初始位置为(0,0,0,N)，表示在原点，方向朝北

示意图如下：



Sprint-I：

- 当Aircraft收到UP指令后，向上移动一个坐标
- 当Aircraft收到DOWN指令后，向下移动一个坐标
- 当Aircraft收到FORWARD指令后，向前移动一个坐标
- 当Aircraft位于地面时(z为0时)，执行DOWN指令无响应

例如:Aircraft位于(0,0,5,N),当收到UP时，新的位置为(0,0,6,N)，继续收到DOWN时，新的位置为(0,0,5,N)，继续收到FORWARD时，新的位置为(0,1,5,N)

Sprint-I Test Case:

```
#include "gtest/gtest.h"
#include "UnmannedAircraft.h"
#include "Instruction.h"

struct UnmannedAircraftTest : testing::Test
{
    void WHEN_I_RECEIVE_INSTRUCTION(const Instruction& instruction)
    {
        aircraft.on(instruction);
```

```
    }

    void THEN_RECEIVE_INSTRUCTION(const Instruction& instruction)
    {
        WHEN_I_RECEIVE_INSTRUCTION(instruction);
    }

    void THE_AIRCRAFT_SHOULD_BE_AT(const Position& position)
    {
        ASSERT_TRUE(position == aircraft.getPosition());
    }

protected:
    UnmannedAircraft aircraft;
};

TEST_F(UnmannedAircraftTest, at_the_beginning_aircraft_should_at_0_0_0_N)
{
    ASSERT_TRUE(Position(0,0,0,N) == aircraft.getPosition());
}

TEST_F(UnmannedAircraftTest, when_receive_instruction_UP_aircraft_should_up_a_step)
{
    WHEN_I_RECEIVE_INSTRUCTION(UP);
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,0,1,N));
}

TEST_F(UnmannedAircraftTest, when_receive_instruction_DOWN_aircraft_should_down_a_step)
{
    WHEN_I_RECEIVE_INSTRUCTION(UP);
    THEN_RECEIVE_INSTRUCTION(DOWN);
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,0,0,N));
}

TEST_F(UnmannedAircraftTest, aircraft_should_not_move_when_receive_instruction_DOWN_on_the_gr
ound)
{
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,0,0,N));
    WHEN_I_RECEIVE_INSTRUCTION(DOWN);
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,0,0,N));
}

TEST_F(UnmannedAircraftTest, when_receive_instruction_FORWARD_aircraft_should_forward_a_step)
{
    WHEN_I_RECEIVE_INSTRUCTION(FORWARD);
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,1,0,N));
}
```

```
//UnmannedAircraft.h
#ifndef _INCL_UNMANNED_AIRCRAFT_H_
#define _INCL_UNMANNED_AIRCRAFT_H_

#include "Position.h"

struct Instruction;
```

```
struct UnmannedAircraft
{
    UnmannedAircraft();

    void on(const Instruction&);

    const Position& getPosition() const;

private:
    Position position;
};

#endif

//UnmannedAircraft.cpp
#include "UnmannedAircraft.h"
#include "Instruction.h"

UnmannedAircraft::UnmannedAircraft() : position(0,0,0,N)
{
}

const Position& UnmannedAircraft::getPosition() const
{
    return position;
}

void UnmannedAircraft::on(const Instruction& instruction)
{
    instruction.exec(position);
}
```

```
//Position.h
#ifndef _INCL_POSITION_H_
#define _INCL_POSITION_H_

enum orientation {N, E, S, W};

struct Position
{
    Position(int x, int y, int z, orientation d);
    bool operator==(const Position& rhs) const;

    void up();
    void down();
    void forward();

private:
    bool onTheGround() const;

private:
    int x;
    int y;
    int z;
    orientation d;
};
```

```
#endif

//Position.cpp
#include "Position.h"

Position::Position(int x, int y, int z, orientation d) : x(x), y(y), z(z), d(d)
{
}

bool Position::operator==(const Position& rhs) const
{
    return x==rhs.x && y==rhs.y && z==rhs.z && d==rhs.d;
}

void Position::up()
{
    ++z;
}

bool Position::onTheGround() const
{
    return z == 0;
}

void Position::down()
{
    if(onTheGround()) return;

    --z;
}

void Position::forward()
{
    ++y;
}
```

```
//Instruction.h
#ifndef _INCL_INSTRUCTION_H_
#define _INCL_INSTRUCTION_H_

struct Position;

struct Instruction
{
    virtual void exec(Position&) const = 0;
    virtual ~Instruction() {}
};

struct Instructions
{
    static Instruction& up();
    static Instruction& down();
    static Instruction& forward();
};
```

```
#define UP Instructions::up()
#define DOWN Instructions::down()
#define FORWARD Instructions::forward()

#endif

//Instruction.cpp
#include "Instruction.h"
#include "Position.h"

namespace
{
    struct UpInstruction : Instruction
    {
    private:
        virtual void exec(Position& position) const
        {
            position.up();
        }
    };
}

Instruction& Instructions::up()
{
    static UpInstruction up;
    return up;
}

namespace
{
    struct DownInstruction : Instruction
    {
    private:
        virtual void exec(Position& position) const
        {
            position.down();
        }
    };
}

Instruction& Instructions::down()
{
    static DownInstruction down;
    return down;
}

namespace
{
    struct ForwardInstruction : Instruction
    {
    private:
        virtual void exec(Position& position) const
        {
            position.forward();
        }
    };
```

```
}

Instruction& Instructions::forward()
{
    static ForwardInstruction forward;
    return forward;
}
```

Sprint-II

- 当Aircraft收到LEFT指令后，向左转90度
- 当Aircraft收到RIGHT指令后，向右转90度
- 当Aircraft收到ROUND指令后，顺时针旋转180度

例如:Aircraft位于(0,0,0,N)，当收到LEFT时，新的位置为(0,0,0,W)，继续收到ROUND，新的位置为(0,0,0,E)，继续收到RIGHT后，新的位置为(0,0,0,S)

```
namespace
{
#define _TEST(scence_description) TEST_F(UnmannedAircraftTest, scence_description)
}
...
_TEST(aircraft_should_turn_to_WEST_when_receive_instruction_LEFT)
{
    WHEN_I_RECEIVE_INSTRUCTION(LEFT);
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,0,0,W));
}

_TEST(aircraft_should_turn_to_SOUTH_when_receive_instruction_LEFT_2_times)
{
    WHEN_I_RECEIVE_INSTRUCTION(LEFT);
    THEN_I_RECEIVE_INSTRUCTION(LEFT);
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,0,0,S));
}

_TEST(aircraft_should_turn_to_EAST_when_receive_instruction_LEFT_3_times)
{
    WHEN_I_RECEIVE_INSTRUCTION(LEFT);
    THEN_I_RECEIVE_INSTRUCTION(LEFT);
    THEN_I_RECEIVE_INSTRUCTION(LEFT);
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,0,0,E));
}

_TEST(aircraft_should_back_to_original_direction_when_receive_instruction_LEFT_4_times)
{
    WHEN_I_RECEIVE_INSTRUCTION(LEFT);
    THEN_I_RECEIVE_INSTRUCTION(LEFT);
    THEN_I_RECEIVE_INSTRUCTION(LEFT);
    THEN_I_RECEIVE_INSTRUCTION(LEFT);
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,0,0,N));
}

_TEST(aircraft_should_turn_to_WEST_when_receive_instruction_RIGHT)
{
```

```
    WHEN_I_RECEIVE_INSTRUCTION(RIGHT);
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,0,0,E));
}

_TEST(aircraft_should_turn_to_SOUTH_when_receive_instruction_RIGHT_2_times)
{
    WHEN_I_RECEIVE_INSTRUCTION(RIGHT);
    THEN_I_RECEIVE_INSTRUCTION(RIGHT);
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,0,0,S));
}

_TEST(aircraft_should_turn_to_EAST_when_receive_instruction_RIGHT_3_times)
{
    WHEN_I_RECEIVE_INSTRUCTION(RIGHT);
    THEN_I_RECEIVE_INSTRUCTION(RIGHT);
    THEN_I_RECEIVE_INSTRUCTION(RIGHT);
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,0,0,W));
}

_TEST(aircraft_should_back_to_original_direction_when_receive_instruction_RIGHT_4_times)
{
    WHEN_I_RECEIVE_INSTRUCTION(RIGHT);
    THEN_I_RECEIVE_INSTRUCTION(RIGHT);
    THEN_I_RECEIVE_INSTRUCTION(RIGHT);
    THEN_I_RECEIVE_INSTRUCTION(RIGHT);
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,0,0,N));
}

_TEST(aircraft_should_turn_to_SOUTH_when_receive_instruction_ROUND)
{
    WHEN_I_RECEIVE_INSTRUCTION(ROUND);
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,0,0,S));
}

_TEST(aircraft_should_back_to_original_direction_when_receive_instruction_ROUND_2_times)
{
    WHEN_I_RECEIVE_INSTRUCTION(ROUND);
    THEN_I_RECEIVE_INSTRUCTION(ROUND);
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,0,0,N));
}

_TEST(aircraft_should_forward_a_step_in_EAST_when_receive_instruction_RIGHT_AND_FORWARD)
{
    WHEN_I_RECEIVE_INSTRUCTION(RIGHT);
    THEN_I_RECEIVE_INSTRUCTION(FORWARD);
    THE_AIRCRAFT_SHOULD_BE_AT(Position(1,0,0,E));
}

_TEST(aircraft_should_forward_a_step_in_WEST_when_receive_instruction_LEFT_AND_FORWARD)
{
    WHEN_I_RECEIVE_INSTRUCTION(LEFT);
    THEN_I_RECEIVE_INSTRUCTION(FORWARD);
    THE_AIRCRAFT_SHOULD_BE_AT(Position(-1,0,0,W));
}

_TEST(aircraft_should_forward_a_step_in_SOUTH_when_receive_instruction_ROUND_AND_FORWARD)
```

```
{
    WHEN_I_RECEIVE_INSTRUCTION(ROUND);
    THEN_I_RECEIVE_INSTRUCTION(FORWARD);
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,-1,0,S));
}
```

```
//Position.h
#ifndef _INCL_POSITION_H_
#define _INCL_POSITION_H_

#include "Coordinate.h"
#include "Orientation.h"

struct Position : Coordinate, Orientation
{
    Position(int x, int y, int z, const Orientation& d);

    bool operator==(const Position& rhs) const;
};

#endif

//Position.cpp
#include "Position.h"

Position::Position(int x, int y, int z, const Orientation& d)
    : Coordinate(x,y,z), Orientation(d)
{
}

bool Position::operator==(const Position& rhs) const
{
    return static_cast<const Coordinate&>(*this) == rhs &&
           static_cast<const Orientation&>(*this) == rhs;
}
```

```
//Coordinate.h
#ifndef _INCL_COORDINATE_H_
#define _INCL_COORDINATE_H_

struct Orientation;

struct Coordinate
{
    Coordinate(int x, int y, int z);

    Coordinate up() const;
    Coordinate down() const;
    Coordinate forward(const Orientation&) const;

    bool operator==(const Coordinate& rhs) const;

private:
    int x;
    int y;
```

```
    int z;
};

#endif

//Coordinate.cpp
#include "Coordinate.h"
#include "Orientation.h"

Coordinate::Coordinate(int x, int y, int z) : x(x), y(y), z(z)
{
}

Coordinate Coordinate::up() const
{
    return Coordinate(x,y,z+1);
}

Coordinate Coordinate::down() const
{
    if(z == 0) return Coordinate(x,y,z);
    return Coordinate(x,y,z-1);
}

Coordinate Coordinate::forward(const Orientation& ori) const
{
    return ori.moveOn(x,y,z);
}

bool Coordinate::operator==(const Coordinate& rhs) const
{
    return x==rhs.x && y==rhs.y && z==rhs.z;
}
```

```
#ifndef _INCL_ORIENTATION_H_
#define _INCL_ORIENTATION_H_

struct Coordinate;

struct Orientation
{
    Orientation turnLeft() const;
    Orientation turnRight() const;
    Orientation turnRound() const;

    Coordinate moveOn(int x, int y, int z) const;

    static const Orientation north;
    static const Orientation east;
    static const Orientation south;
    static const Orientation west;

    bool operator==(const Orientation&) const;

private:
        Orientation(int order, int xFactor, int yFactor);
```

```
private:
    int order;
    int xFactor;
    int yFactor;
};

#define N Orientation::north
#define E Orientation::east
#define S Orientation::south
#define W Orientation::west

#endif

//Orientation.cpp
#include "Orientation.h"
#include "Coordinate.h"

namespace
{
    Orientation* orientations[4];
}

Orientation::Orientation(int order, int xFactor, int yFactor)
    : order(order), xFactor(xFactor), yFactor(yFactor)
{
    orientations[order] = this;
}

Orientation Orientation::turnLeft() const
{
    return *orientations[(order+3)%4];
}

Orientation Orientation::turnRight() const
{
    return *orientations[(order+1)%4];
}

Orientation Orientation::turnRound() const
{
    return *orientations[(order+2)%4];
}

bool Orientation::operator==(const Orientation& rhs) const
{
    return order == rhs.order;
}

Coordinate Orientation::moveOn(int x, int y, int z) const
{
    return Coordinate(x+xFactor,y+yFactor,z);
}

const Orientation Orientation::north(0,  0,  1);
const Orientation Orientation::east( 1,  1,  0);
const Orientation Orientation::south(2,  0, -1);
```

```
const Orientation Orientation::west( 3, -1,  0);
```

```
//Instruction.h
#ifndef _INCL_INSTRUCTION_H_
#define _INCL_INSTRUCTION_H_

struct Coordinate;
struct Orientation;

struct Instruction
{
    virtual void exec(Coordinate&, Orientation&) const = 0;
    virtual ~Instruction() {}
};

struct Instructions
{
    static Instruction& up();
    static Instruction& down();
    static Instruction& forward();
    static Instruction& left();
    static Instruction& right();
    static Instruction& round();
};

#define UP Instructions::up()
#define DOWN Instructions::down()
#define FORWARD Instructions::forward()
#define LEFT Instructions::left()
#define RIGHT Instructions::right()
#define ROUND Instructions::round()

#endif

//Instruction.cpp
#include "Instruction.h"
#include "Coordinate.h"
#include "Orientation.h"

namespace
{
    struct UpInstruction : Instruction
    {
    private:
        virtual void exec(Coordinate& coor, Orientation&) const
        {
            coor = coor.up();
        }
    };
}

Instruction& Instructions::up()
{
    static UpInstruction up;
    return up;
}
```

```
namespace
{
    struct DownInstruction : Instruction
    {
    private:
        virtual void exec(Coordinate& coor, Orientation&) const
        {
            coor = coor.down();
        }
    };
}

Instruction& Instructions::down()
{
    static DownInstruction down;
    return down;
}

namespace
{
    struct ForwardInstruction : Instruction
    {
    private:
        virtual void exec(Coordinate& coor, Orientation& ori) const
        {
            coor = coor.forward(ori);
        }
    };
}

Instruction& Instructions::forward()
{
    static ForwardInstruction forward;
    return forward;
}

namespace
{
    struct LeftInstruction : Instruction
    {
    private:
        virtual void exec(Coordinate&, Orientation& ori) const
        {
            ori = ori.turnLeft();
        }
    };
}

Instruction& Instructions::left()
{
    static LeftInstruction left;
    return left;
}

namespace
```

```
{
    struct RightInstruction : Instruction
    {
    private:
        virtual void exec(Coordinate&, Orientation& ori) const
        {
            ori = ori.turnRight();
        }
    };

}

Instruction& Instructions::right()
{
    static RightInstruction right;
    return right;
}

namespace
{
    struct RoundInstruction : Instruction
    {
    private:
        virtual void exec(Coordinate&, Orientation& ori) const
        {
            ori = ori.turnRound();
        }
    };

}

Instruction& Instructions::round()
{
    static RoundInstruction round;
    return round;
}
```

Sprint-III

- 当Aircraft收到FORWARD_N(n)指令后，向前移动n个坐标
- 当Aircraft收到UP_N(n)指令后，向上移动n个坐标
- 当Aircraft收到DOWN_N(n)指令后，向下移动n个坐标

例如:Aircraft位于(0,0,0,N)，当收到FORWARD_N(10)，新的位置为(0,10,0,N)，继续收到UP_N(10)，新的位置为(0,10,10,N)，继续收到DOWN_N(5)，新的位置为(0,10,5,N)

```
_TEST(aircraft_should_forward_n_step_when_receive_instruction_FORWARD_N)
{
    WHEN_AIRCRAFT_EXECUTE_INSTRUCTION(FORWARD_N(10));
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,10,0,N));
}

_TEST(aircraft_should_up_n_step_when_receive_instruction_UP_N)
```

```
{
    WHEN_AIRCRAFT_EXECUTE_INSTRUCTION(UP_N(10));
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,0,10,N));
}

_TEST(aircraft_should_down_n_step_when_receive_instruction_DOWN_N)
{
    WHEN_AIRCRAFT_EXECUTE_INSTRUCTION(UP_N(10));
    THEN_AIRCRAFT_EXECUTE_INSTRUCTION(DOWN_N(10));
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,0,0,N));
}
```

```
//Instruction.h
    ...
struct Instructions
{
    ...
    static Instruction& repeat(const Instruction&, int n);
};

#define REPEAT(instruction, n) Instructions::repeat(instruction, n)
...
#define FORWARD_N(n) REPEAT(FORWARD, n)
#define UP_N(n) REPEAT(UP, n)
#define DOWN_N(n) REPEAT(DOWN, n)

#endif

//Instruction.cpp
...
namespace
{
    struct RepeatInstruction : Instruction
    {
        RepeatInstruction() : ins(0), num(0)
        {
        }

        void update(const Instruction& _ins, int n)
        {
            ins = &_ins;
            num = n;
        }

    private:
        virtual void exec(Coordinate& coor, Orientation& ori) const
        {
            for(int i = 0; i < num; ++i)
            {
                ins->exec(coor, ori);
            }
        }

    private:
        const Instruction* ins;
        int num;
```

```
    };
}

Instruction& Instructions::repeat(const Instruction& ins, int n)
{
    static RepeatInstruction repeat;
    repeat.update(ins,n);
    return repeat;
}
```

Sprint-IV

- 当Aircraft收到REPEAT(instruction,n)指令后，循环执行instruction指令n次
- instruction指令是除REPEAT指令之外的任意指令，n的范围为[0,10]

例如:Aircraft位于(0,0,0,N)，收到REPEAT(FORWARD,5)，新的位置为(0,5,0,N)，继续收到REPEAT(LEFT,1)，新的位置为(0,5,0,W)，继续收到REPEAT(FORWARD,5)，新的位置为(-5,5,0,W)，继续收到REPEAT(UP,5)，新的位置为(-5,5,5,W)

```
_TEST(aircraft_should_execute_instructions_repeative_when_receive_REPEAT_INSTRUCTION_N)
{
    WHEN_AIRCRAFT_EXECUTE_INSTRUCTION(REPEAT(FORWARD,5));
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,5,0,N));

    THEN_AIRCRAFT_EXECUTE_INSTRUCTION(REPEAT(LEFT,1));
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,5,0,W));

    THEN_AIRCRAFT_EXECUTE_INSTRUCTION(REPEAT(FORWARD,5));
    THE_AIRCRAFT_SHOULD_BE_AT(Position(-5,5,0,W));

    THEN_AIRCRAFT_EXECUTE_INSTRUCTION(REPEAT(UP,5));
    THE_AIRCRAFT_SHOULD_BE_AT(Position(-5,5,5,W));
}

_TEST(aircraft_should_stay_on_original_position_when_receive_repeat_n_out_of_bound)
{
    WHEN_AIRCRAFT_EXECUTE_INSTRUCTION(REPEAT(UP,11));
    THE_AIRCRAFT_SHOULD_BE_AT(Position(0,0,0,N));
}
```

```
//Instruction.h
struct RepeatableInstruction : Instruction
{
    RepeatableInstruction(const Instruction&, int n);
private:
    virtual void exec(Coordinate&, Orientation&) const;
    bool isOutOfBound() const;
private:
    const Instruction& ins;
    const int n;
};

#define REPEAT(instruction, n) RepeatableInstruction(instruction, n)

//Instruction.cpp
```

```
epeatableInstruction::RepeatableInstruction(const Instruction& ins, int n)
    : ins(ins), n(n)
{
}

bool RepeatableInstruction::isOutOfBound() const
{
    return n<0 || n>10;
}

void RepeatableInstruction::exec(Coordinate& coor, Orientation& ori) const
{
    if(isOutOfBound()) return;

    for(int i = 0; i < n; ++i)
    {
        ins.exec(coor, ori);
    }
}
```

Refactor

```
//UnmannedAircraft.h
#ifndef _INCL_UNMANNED_AIRCRAFT_H_
#define _INCL_UNMANNED_AIRCRAFT_H_

#include "Position.h"

struct Instruction;

struct UnmannedAircraft
{
    UnmannedAircraft();
    void on(const Instruction&);
    const Position& getPosition() const;

private:
    Position position;
};

#endif

//UnmannedAircraft.cpp
#include "UnmannedAircraft.h"
#include "Instruction.h"

UnmannedAircraft::UnmannedAircraft() : position(0,0,0,N)
{
}

const Position& UnmannedAircraft::getPosition() const
{
    return position;
}
```

```
void UnmannedAircraft::on(const Instruction& instruction)
{
    instruction.exec(position.ROLE(Coordinate), position.ROLE(Orientation));
}
```

```
//Position.h
#ifndef _INCL_POSITION_H_
#define _INCL_POSITION_H_

#include "Coordinate.h"
#include "Orientation.h"
#include "base/Role.h"

struct Position : Coordinate, Orientation
{
    Position(int x, int y, int z, const Orientation& d);
    bool operator==(const Position& rhs) const;

    IMPL_ROLE(Coordinate);
    IMPL_ROLE(Orientation);
};

#endif

//Position.cpp
#include "Position.h"

Position::Position(int x, int y, int z, const Orientation& d)
    : Coordinate(x,y,z), Orientation(d)
{
}

bool Position::operator==(const Position& rhs) const
{
    return ROLE(Coordinate) == rhs.ROLE(Coordinate) &&
           ROLE(Orientation) == rhs.ROLE(Orientation);
}
```

```
//Coordinate.h
#ifndef _INCL_COORDINATE_H_
#define _INCL_COORDINATE_H_

struct Orientation;

struct Coordinate
{
    Coordinate(int x, int y, int z);

    Coordinate up() const;
    Coordinate down() const;
    Coordinate forward(const Orientation&) const;
    bool operator==(const Coordinate& rhs) const;

private:
```

```
    int x;
    int y;
    int z;
};

#endif

//Coordinate.cpp
#include "Coordinate.h"
#include "Orientation.h"

Coordinate::Coordinate(int x, int y, int z) : x(x), y(y), z(z)
{
}

Coordinate Coordinate::up() const
{
    return Coordinate(x,y,z+1);
}

namespace
{
    bool onGround(int z)
    {
        return z == 0;
    }
}

Coordinate Coordinate::down() const
{
    if(onGround(z)) return Coordinate(x,y,z);

    return Coordinate(x,y,z-1);
}

Coordinate Coordinate::forward(const Orientation& ori) const
{
    return ori.moveOn(x,y,z);
}

bool Coordinate::operator==(const Coordinate& rhs) const
{
    return x==rhs.x && y==rhs.y && z==rhs.z;
}
```

```
//Orientation.h
#ifndef _INCL_ORIENTATION_H_
#define _INCL_ORIENTATION_H_

struct Coordinate;

struct Orientation
{
    Orientation turnLeft() const;
    Coordinate moveOn(int x, int y, int z) const;
    bool operator==(const Orientation&) const;
```

```
    static const Orientation north;
    static const Orientation east;
    static const Orientation south;
    static const Orientation west;

private:
        Orientation(int order, int xFactor, int yFactor);
private:
    int order;
    int xFactor;
    int yFactor;
};

#define N Orientation::north
#define E Orientation::east
#define S Orientation::south
#define W Orientation::west

#endif

//Orientation.cpp
#include "Orientation.h"
#include "Coordinate.h"

namespace
{
    Orientation* orientations[4];
}

Orientation::Orientation(int order, int xFactor, int yFactor)
    : order(order), xFactor(xFactor), yFactor(yFactor)
{
    orientations[order] = this;
}

Orientation Orientation::turnLeft() const
{
    return *orientations[(order+3)%4];
}

bool Orientation::operator==(const Orientation& rhs) const
{
    return order == rhs.order;
}

Coordinate Orientation::moveOn(int x, int y, int z) const
{
    return Coordinate(x+xFactor,y+yFactor,z);
}

const Orientation Orientation::north(0,  0,  1);
const Orientation Orientation::east( 1,  1,  0);
const Orientation Orientation::south(2,  0, -1);
const Orientation Orientation::west( 3, -1,  0);
```

```
//Instruction.h
#ifndef _INCL_INSTRUCTION_H_
#define _INCL_INSTRUCTION_H_

struct Coordinate;
struct Orientation;

struct Instruction
{
    virtual void exec(Coordinate&, Orientation&) const = 0;
    virtual ~Instruction() {}
};

struct Instructions
{
    static Instruction& up();
    static Instruction& down();
    static Instruction& forward();
    static Instruction& left();
};

struct RepeatableInstruction : Instruction
{
    RepeatableInstruction(const Instruction&, int n);
private:
    virtual void exec(Coordinate&, Orientation&) const;
    bool isOutOfBound() const;
private:
    const Instruction& ins;
    const int n;
};

#define REPEAT(instruction, n) RepeatableInstruction(instruction, n)
#define UP Instructions::up()
#define DOWN Instructions::down()
#define FORWARD Instructions::forward()
#define LEFT Instructions::left()
#define RIGHT REPEAT(LEFT, 3)
#define ROUND REPEAT(LEFT, 2)
#define FORWARD_N(n) REPEAT(FORWARD, n)
#define UP_N(n) REPEAT(UP, n)
#define DOWN_N(n) REPEAT(DOWN, n)

#endif

//Instruction.cpp
#include "Instruction.h"
#include "Coordinate.h"
#include "Orientation.h"
#include "base/Singleton.h"
#include "base/Keywords.h"

Instruction& Instructions::up()
{
    DEF_SINGLETON(UpInstruction) EXTENDS(Instruction)
    {
```

```
    private:
        virtual void exec(Coordinate& coor, Orientation&) const
        {
            coor = coor.up();
        }
    };

    return UpInstruction::getInstance();
}

Instruction& Instructions::down()
{
    DEF_SINGLETON(DownInstruction) EXTENDS(Instruction)
    {
    private:
        virtual void exec(Coordinate& coor, Orientation&) const
        {
            coor = coor.down();
        }
    };

    return DownInstruction::getInstance();
}

Instruction& Instructions::forward()
{
    DEF_SINGLETON(ForwardInstruction) EXTENDS(Instruction)
    {
    private:
        virtual void exec(Coordinate& coor, Orientation& ori) const
        {
            coor = coor.forward(ori);
        }
    };

    return ForwardInstruction::getInstance();
}

Instruction& Instructions::left()
{
    DEF_SINGLETON(LeftInstruction) EXTENDS(Instruction)
    {
    private:
        virtual void exec(Coordinate&, Orientation& ori) const
        {
            ori = ori.turnLeft();
        }
    };

    return LeftInstruction::getInstance();
}

RepeatableInstruction::RepeatableInstruction(const Instruction& ins, int n)
    : ins(ins), n(n)
{
}
```

```
bool RepeatableInstruction::isOutOfBound() const
{
    return n<0 || n>10;
}

void RepeatableInstruction::exec(Coordinate& coor, Orientation& ori) const
{
    if(isOutOfBound()) return;

    for(int i = 0; i < n; ++i)
    {
        ins.exec(coor, ori);
    }
}
```

github source codes : https://github.com/liyongshun/unmanned-aircraft-tdd