I.T.U.

Faculty of Electric-Electronic

Computer Engineering

Lesson name: Object Oriented Programming

Lesson Code: BLG252E

Name Surname: Abdullah AYDEĞER

Number: 040090533

Instructor's Name: Sanem SARIEL TALAY

Due Date: 26.05.2011

What This Report Includes?

- Introduction
- Classes
- UML Diagrams

Introduction

I've used Microsoft Visual Studio compiler to compile my codes. I wrote class for solving the game with recursion. But I have some run-time error which I can't understand why they exist. In my homework, I add all people to id's and control all states with their id's. Therefore, person names can be changed or can be entered by user. In Container I added people with their id part. For example; Father id = 0 and first element of generic array is Father. But missing part of my homework, Son1 and Son2 has the same id, and so on program is trying to add two different people(in same class) in the same memory location. Consequently, for some situations, my codes can't work properly.

Classes

- **Game**

All methods and members can be shown in the UML diagram (last page of the report). Necessary comments are on the code can be seen in the figure.

This class is used for controlling inconsistencies while solving game with recursion (letsTry function).

```
class Game{
    private:
        Container<Person> left,right;   //Containers hold people which are on the right and which are on the left
        int counter1,counter2;          //These are counters for recursion
        bool placer;                    //placer determines the raft place
    public:
        Game();
        void addToLeft(Person );        //For add one Person to Left Container
        void addToRight(Person );       //For add one Person to Right Container
        Person takeFromLeft();          //Taking one Person from Left Container
        Person takeFromRight();         //Taking one Person from Right Container
        bool isFinished(Container<Operator>, Container<NonOperator> ) const;    //This function controls the game is finished or not
        bool sailAcrossFromLeftToRight(Person &, Person &);
        //bool sailAcrossLeftToRight(Operator &);          //This function does not needed while solving the game
        bool sailAcrossFromRightToLeft(Person &, Person&);
        bool sailAcrossRightToLeft(Person &);
        bool isConsistency(Container<Person> &) const;      //This function controls all the inconsistencies
        bool isHere(int, Container<Person> ) const;         //This function helps the isConsistency function
        bool letsTry(Container<Operator> &, Container<NonOperator> &);  //This function is the base function for trying the solution
};
#endif
```

- **Container**

All methods and members can be shown in the UML diagram (last page of the report). Necessary comments are on the code can be seen in the figure.

This class is needed for holding data in containers. In this design containers are generic arrays.

```
template <class Type>
    class Container{
        private:
            Type *dizi;                 //This is generic array
            int numberOfElements;       //This holds the number of elements of generic array
        public:
            Container(){
                dizi = new Type[200];   //Default constructor allocates 20 member for generic array
                numberOfElements = 0;}

            Container(int number){
                dizi = new Type[5*number];  //Constructor allocates 2*number member for generic array
                numberOfElements = 0;}

            void addToArray(Type addType){
                dizi[addType.getID()] = addType;    //In this line, adding the new member to generic array according to new member id
                numberOfElements++;}        //Incrementing the number of elements of generic array
```

```
        Type Container<Type>::takeFromArray(int number){
            Type t = dizi[0];
            if(number>6 || number<0){       //If passing integer is not valid
                cout<<"WRONG NUMBER\n";     //Then cout wrong number and
                return t;}                  //return the first element of generic array
                    t = dizi[number];
                    Person p = Person();
                    dizi[number] = p;   //For taking array, adding new Person to this partition
                    numberOfElements--;     //Decreasing the number of elemets of generic array
            return t;}

        Type getArray(int number){      //This function get one element from generic array according to integer parameter
            Type t = dizi[0];
            if(number>6 ||number<0){
                cout<<"WRONG NUMBER\n";
                return t;}
            return dizi[number];}

        int getnumberOfElements() const{
            return numberOfElements;}
};
```

- **Person**

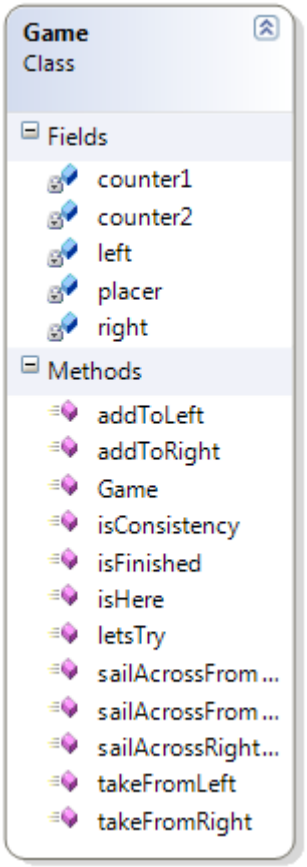
All methods and members can be shown in the UML diagram (last page of the report). Necessary comments are on the code can be seen in the figure.

This class is used for showing all people's requirements.

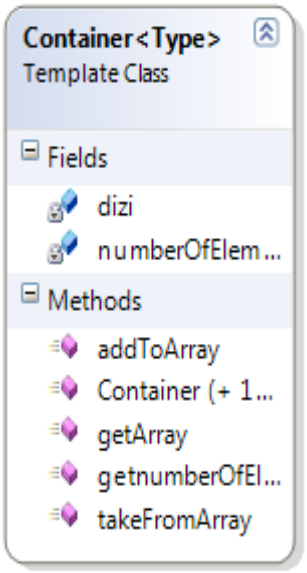
```
class Person{      //Base class for all people
    protected:
        int id;         //All people have id (in my design)
        string name;    //All people absolutely have name
        string place;   //All people should hold their place (right or left)
    public:
        Person(){ name=" "; id = -1; };
        Person(string );
        int getID() const;
        string getName() const;
        void setPlace(string );
        string getPlace()const;
        virtual bool canGetOnTheRaftWith(const Person &) const { cout<<"\n\nPersondayız\n\n";  return false;}   //This is virtual function since all Operators have these function
};

class Operator:public Person{
    protected:          //Does not need any protected or private members
    public:
        Operator(){};
        Operator(string );
        virtual bool canGetOnTheRaftWith(const Person &) const { cout<<"\n\nOperatordeyizz\n\n";  return false;}
};
```

```
//NonOperators inherited class only have constructor in their class
class Son:public NonOperator{
    public:
        Son(string );
};

class Daughter:public NonOperator{
    public:
        Daughter(string );
};

class Thief:public NonOperator{
    public:
        Thief(string );
};
```

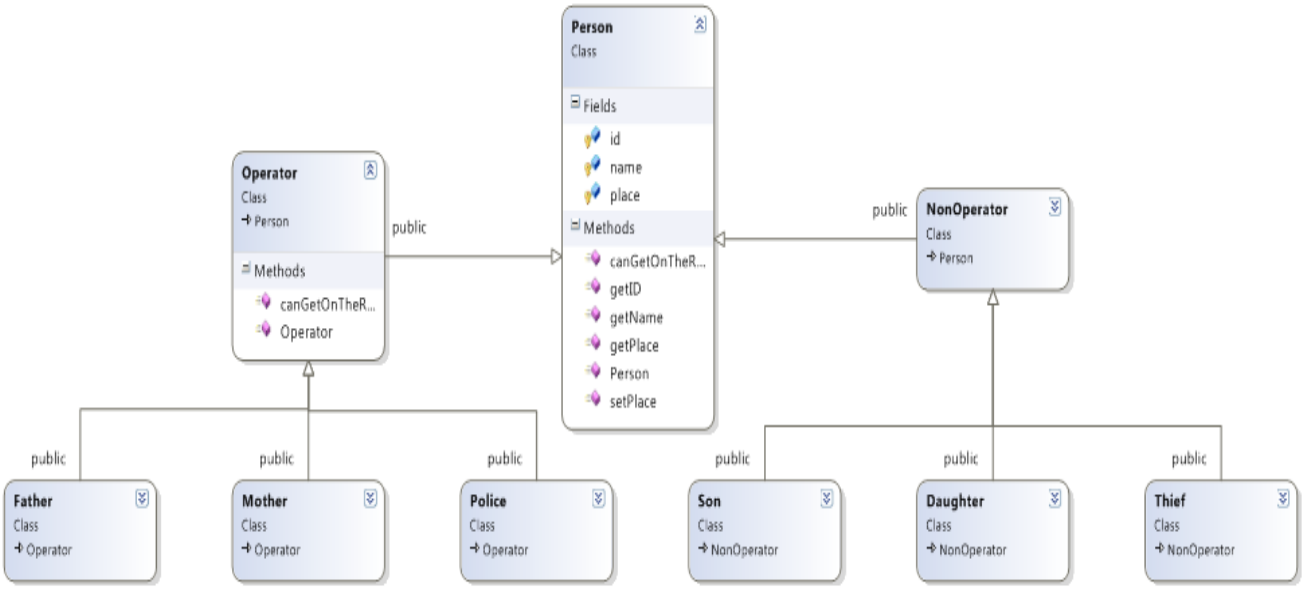
UML DIAGRAMS



Game



Container



Person