C++ to all

C++ and OOP

Part 1

Contents

Introduction

Class is a data type, template that is used to model some abstract or concrete concept.

Object is a variable.

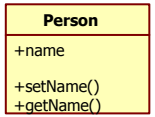
How to model a class?
UML is used.

Unified Modeling Language

UML-tools
StarUML, argoUML, Ms Visio

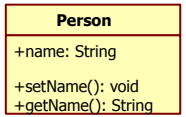
Example: Person class

Version 1



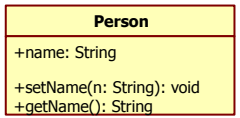
Version 2

-



Version 3

-



Access specifiers.

+ public

- private

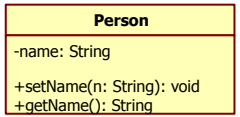
protected

Main rule:
Attributes are private

Methods are public

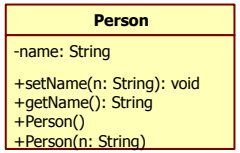
Version 4

-



Final version ☺

-



Code

```
class Person {
    private:
    string name;
    public:
     void setName(string n) {
       name = n;
    }

    string getName() {
       return name;
    }

    Person() {

    }

     Person(string n) {
       name = n;
    }
};
```

Test run

```
Car car1;
car1.setData("Ford", 2019);
cout << "\nManufacturer is " << car1.getManuf() << endl;
Person p1;
p1.setName("Darry");
cout << p1.getName() << endl;

car1.addOwner(p1);
cout << "\nOwner is "<< car1.getOwnerInfo() << endl;
```

Output

```
Darry
```

Exercise/Example

Model and code class **Car** with 2 attributes, constructors and methods.

```
class Car {
    private:
     string manufacturer;
     int yearModel;
    public:
        Car() {
    }
     Car(string n) {
       manufacturer = n;
    }
     Car(string n, int y) {
       manufacturer = n;
       yearModel = y;
    }
     void setData(string n, int y) {
       manufacturer = n;
       yearModel = y;
    }
    string getManuf() {
       return manufacturer;
    }
    int getYearModel() {
       return yearModel;
    }
};
```

Test run

```
int main()
{
    Car car1;
    car1.setData("Ford", 2019);
    cout << "\nManufacturer is " << car1.getManuf() << endl;
```

Output

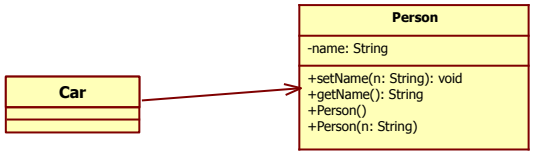
```
Manufacturer is Ford
```

Relationships between objects

Association

Example

Car has an owner.



Code

```
class Car {
    private:
     string manufacturer;
     int yearModel;
     Person owner;
    public:
        Car() {
    }
     Car(string n) {
       manufacturer = n;
    }
     Car(string n, int y) {
       manufacturer = n;
       yearModel = y;
    }
    void addOwner (Person p)
    {
        owner = p;
    }
    string getOwnerInfo()
    {
        return owner.getName();
    }
     void setData(string n, int y) {
       manufacturer = n;
       yearModel = y;
    }
    string getManuf() {
       return manufacturer;
    }
    int getYearModel() {
       return yearModel;
    }
};
```

Person class has not changed

```
class Person {
    private:
    string name;
    public:
     void setName(string n) {
        name = n;
    }

    string getName() {
        return name;
    }

    Person() {

    }

     Person(string n) {
        name = n;
    }
};
```

Test run

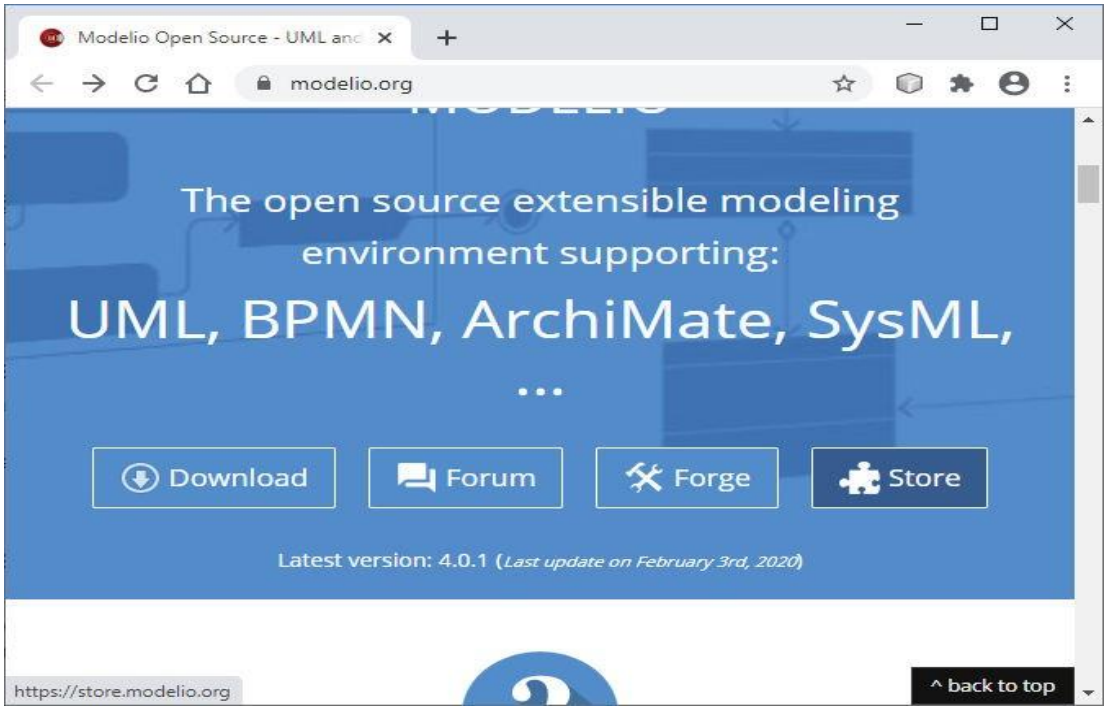
```
int main()
{
    Car car1;
    car1.setData("Ford", 2019);
    cout << "\nManufacturer is " << car1.getManuf() << endl;
    Person p1;
    p1.setName("Darry");
    cout << p1.getName() << endl;

    car1.addOwner(p1);
    cout << "\nOwner is "<< car1.getOwnerInfo() << endl;
}
```

Output

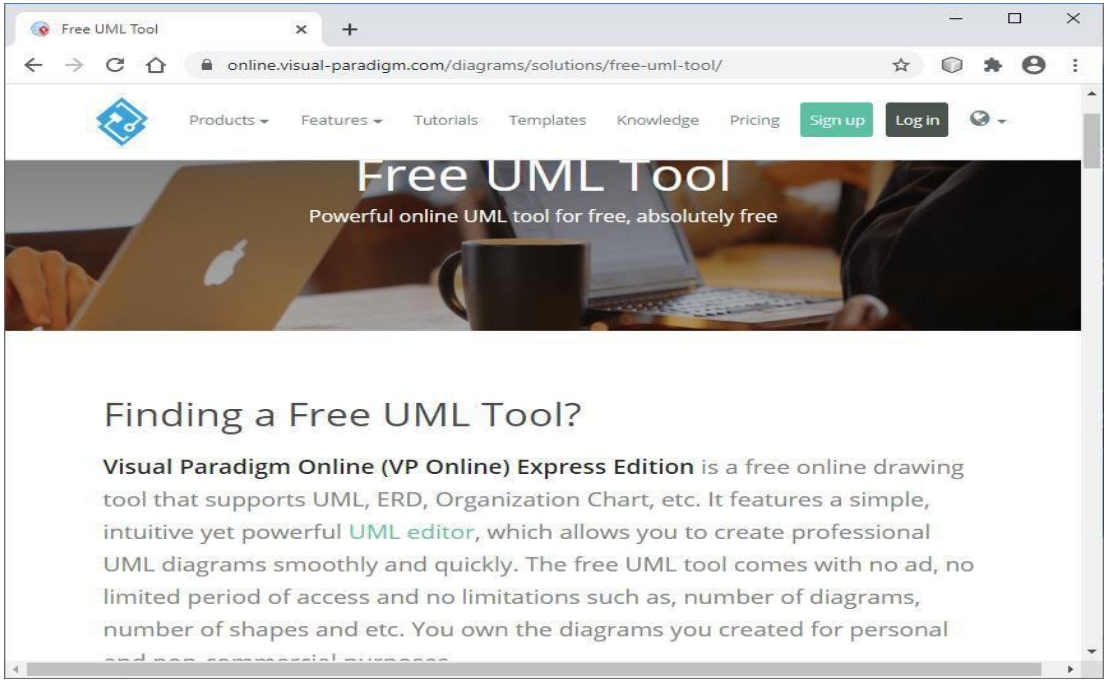
```
Manufacturer is Ford
Darry

Owner is Darry
```

What UML tool to use?

Here is one possible



There are tools like ArgoUML, Umlet, StarUML, Ms Vision and also online tools like VP:



When class diagram gets more complicated, UML tool helps a lot. AND at the same time you can create a document (specification, model) about static structure of the system.

Example: Car Rental Application

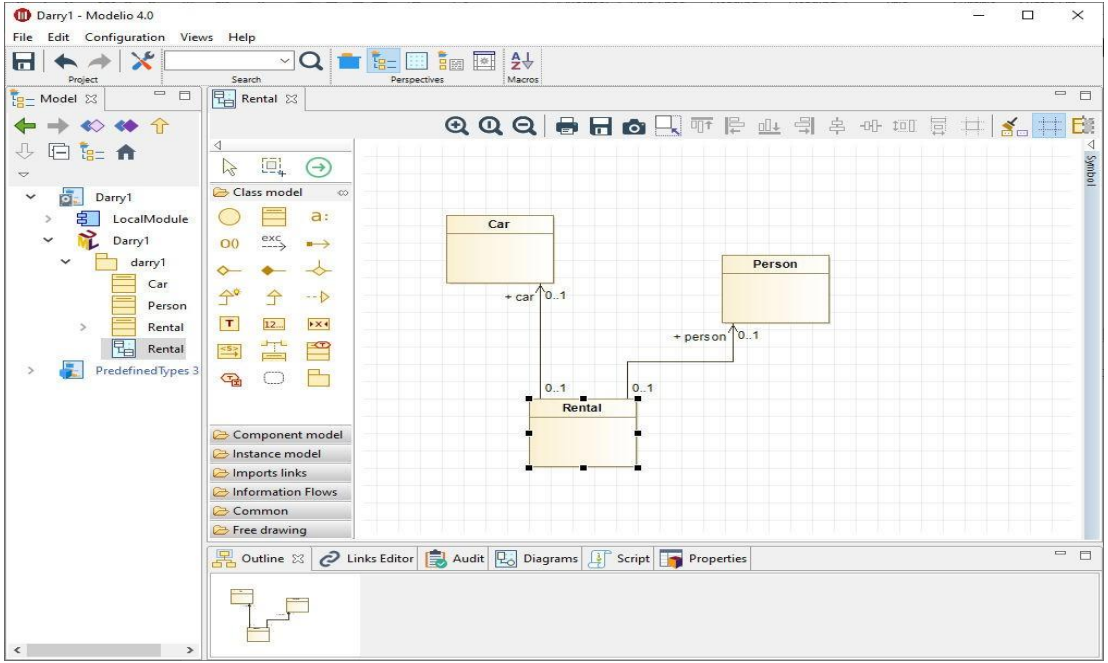
Customer can rent a car and rental contract is done.

We need to create objects like

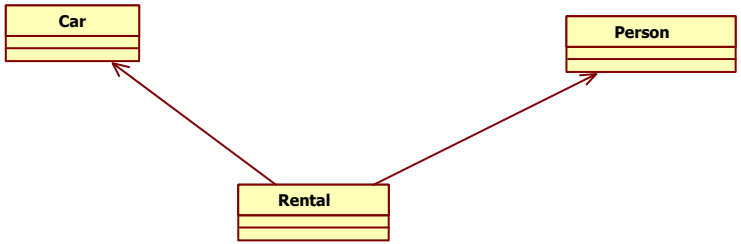
Customer
Car Rental

They are to be modelled in a class diagram.

Here was Modelio used to create the basic class diagram:


Darry1 - Modelio 4.0
File Edit Configuration Views Help
Project
Search
Perspectives
Macros
Model
Darry1
LocalModule
Darry1
darry1
Car
Person
Rental
Rental
PredefinedTypes 3
Rental
Class model
Component model
Instance model
Imports links
Information Flows
Common
Free drawing
Car
Person
Rental
+ car
0..1
+ person
0..1
0..1
0..1
Symbol
Outline
Links Editor
Audit
Diagrams
Script
Properties

Class diagram



Codes

```
class Person {
    private:
    string name;
    public:
     void setName(string n) {
       name = n;
    }

    string getName() {
       return name;
    }

    Person() {

    }

     Person(string n) {
       name = n;
    }
};
```

```
class Car {
    private:
     string manufacturer;
     int yearModel;
     double  pricePerDay;
    public:
        Car() {
    }
    Car(string n) {
       manufacturer = n;
    }
     Car(string n, int y) {
       manufacturer = n;
       yearModel = y;
    }
     void setData(string n, int y) {
       manufacturer = n;
       yearModel = y;
    }
    string getManuf() {
       return manufacturer;
    }
     int getYearModel() {
            return yearModel;
    }
    double getPricePerDay()
    {

        return pricePerDay;
    }
    void setPricePerDay(double p)
    {
        pricePerDay = p;
    }
```

```
class Rental
{
    private:
        int days;
        Car rentedCar;
        Person customer;

    public:
    void createContract(int d, Car c, Person p)
    {
        days = d;
        rentedCar = c;
        customer = p;
    }

    string getAgreement()
    {
        string contract = "The rented car is " + rentedCar.getManuf();
        contract += ". Customer is " + customer.getName();
        double totalPrice = days * rentedCar.getPricePerDay();
        contract += ". The price is " + to_string(totalPrice);

        return contract;
    }

};
```

Test run

```
int main()
{
    Car car1;
    car1.setData("Ford", 2019);
    car1.setPricePerDay(66);
    cout << "\nManufacturer is " << car1.getManuf() << endl;
    Person p1;
    p1.setName("Darry");
    cout << p1.getName() << endl;

    Rental rent;
    rent.createContract(5, car1, p1);
    cout << rent.getAgreement();
}
```

Output

```
Manufacturer is Ford
Darry
The rented car is Ford. Customer is Darry. The price is 330.000000
```

Part 1 END

C++ to all

C++ and classes

Part 2

Aggregation

A class contains 1 .. n other classes (we could speak about objects …)

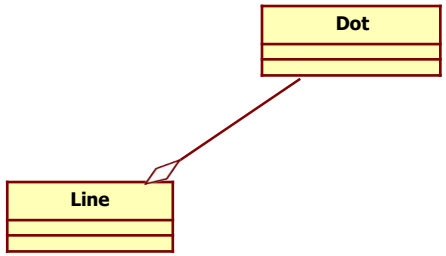
E.g.

A team has 3 members.

A line has a starting point and an ending point.

Cowhouse is meant for 20 cows.

Example: Line has 2 points.



Code

```
class Dot {
    private:
        int x;
        int y;
    public:
        Dot()
        {
        }
        Dot(int xx, int yy)
        {
            x = xx;
            y = yy;
        }
        void setDot(int xx, int yy)
        {
            x = xx;
            y = yy;
        }
        int getX()
        {
            return x;
        }
        int getY()
        {
            return y;
        }
        string dotInfo()
        {
            string info = "("  + to_string(x) + "," + to_string(y) + ")";
            return info;
        }

};
```

```
class Line {
    private:
        double thickness;
        Dot startingPoint;
        Dot endingPoint;
    public:
        Line()
        {
        }

        Line(Dot a, Dot b, double t)
        {
            startingPoint = a;
            endingPoint = b;
            thickness = t;
        }

    // other methods could be added of course here ...

        string getInfo()
        {
            string info = "The thickness is " + to_string(thickness);
            info = info + "\nand starting point is " + startingPoint.dotInfo();
            info = info + "\nand ending point is " + endingPoint.dotInfo();
            return info;
        }


};
```

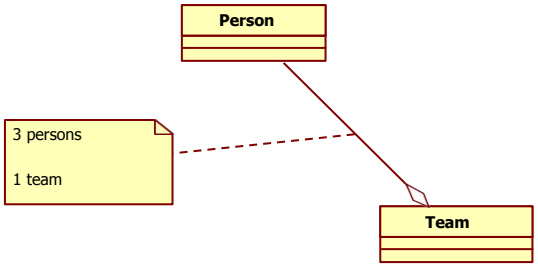
Test run

```
int main()
{
    Dot d1;
    Dot d2;
    d1.setDot(5,6);
    d2.setDot(10,12);
    cout << d1.dotInfo() << endl;
    cout << d1.dotInfo() << endl;

    Line line1(d1,d2, 2.5);
    cout << line1.getInfo();

}
```

Output

```
(5,6)
(5,6)
The thickness is 2.500000
and starting point is (5,6)
and ending point is (10,12)
```

Exercise/Example: Team has 3 members.



Solution

```
class Person {
    private:
        string name;
    public:
        void setName(string n) {
            name = n;
        }

        string getName() {
            return name;
        }

        Person() {
        }

         Person(string n) {
            name = n;
        }
};
```

```
class Team {
    private:
        string teamName;
        Person member1;
        Person member2;
        Person member3;
    public:
        Team()
        {}
        Team(string name)
        {
          teamName = name;
        }
        Team(string name, Person p1, Person p2, Person p3)
        {
          teamName = name;
          member1 = p1;member2 = p2;member3 = p3;
        }
        void setData(string name, Person p1, Person p2, Person p3)
        {
          teamName = name;
          member1 = p1;member2 = p2;member3 = p3;
        }
        string getInfo()
        {
            string info = "Team's name is " + teamName;
            info = info + " and members are: " + member1.getName();
            info = info + ", " + member2.getName();
            info = info + " and " + member3.getName();
            return info;
        }
};
```

Test run

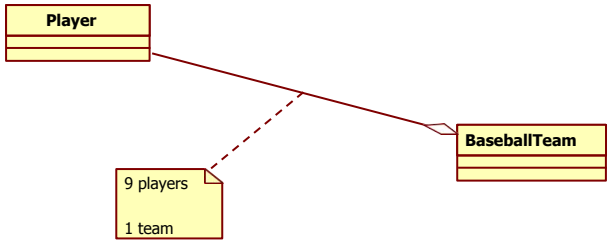
```
int main()
{
    Person p1;
    p1.setName("Darry");
    cout << p1.getName() << endl;
    Person p2;
    p2.setName("Harry");
    cout << p2.getName() << endl;
    Person p3;
    p3.setName("Larry");
    cout << p3.getName() << endl;

    Team trio;
    trio.setData("Piccolos", p1, p2, p3);
    cout << trio.getInfo();
}
```

Output

```
Darry
Harry
Larry
Team's name is Piccolos and members are: Darry, Harry and Larry
```

Example: A baseball team has 9 players.

We need an array of person references.



Codes

```
class Player {
    private:
        string name;
    public:
        void setName(string n) {
            name = n;
        }

        string getName() {
            return name;
        }

        Player() {
        }

        Player(string n) {
            name = n;
        }
};
```

```
class BaseballTeam {
    private:
        string teamName;
        Player players[9];
    public:
        BaseballTeam()
        {}
        BaseballTeam(string name)
        {
          teamName = name;
        }

        void setData(Player list[])
        {
          for (int i = 0; i < 9; i++)
            players[i] = list[i];
        }

        string getInfo()
        {
            string info = "Team's name is " + teamName;
            info = info + " and members are: \n";
            for (int i = 0; i < 9; i++)
              info += players[i].getName() + "\n";
            return info;
        }
};
```

Test run

```
int main()
{
    BaseballTeam team1("NY");

    Player teamPlayers[9];
    string names[] = {"Kim","Tim","Jim", "Bob", "Rick", "Ted", "Lee", "Dan", "Don"};
    for (int i = 0; i < 9; i++)
     teamPlayers[i]  = Player(names[i]);

     team1.setData(teamPlayers);

     cout << team1.getInfo();

}
```

Output

```
Team's name is NY and members are:
Kim
Tim
Jim
Bob
Rick
Ted
Lee
Dan
Don
```

Note: we have used only one way to add players to the team. There are of course several ways: everything depends on the needs..

Exercise/Example: A polygon contains max 10 corner points.

Create classes, constructors and methods.

Note:

you can use random number generator when creating points (x-/y-coordinates).

Now we again use a fixed array to hold polygon corner points.

But in real world you should already consider a dynamic data structure for dots.

Codes

```
class Dot {
    private:
        int x;
        int y;
    public:
        Dot()
        {
        }
        Dot(int xx, int yy)
        {
            x = xx;
            y = yy;
        }
        void setDot(int xx, int yy)
        {
            x = xx;
            y = yy;
        }
        int getX()
        {
            return x;
        }
        int getY()
        {
            return y;
        }
        string dotInfo()
        {
            string info = "("  + to_string(x) + "," + to_string(y) + ")";
            return info;
        }

};
```

```
class Polygon {
    private:
        Dot corners[10];

    public:
        Polygon()
        {
        }

        void addCornerPoints()
        {
            for (int k = 0; k < 10; k++)
            {
                int xx = rand() % 50;
                int yy = rand() % 50;
                corners[k] = Dot(xx, yy);
            }
        }

        string getCorners()
        {
            string info = "Corner points: \n";
            for (int k = 0; k < 10; k++)
                info += corners[k].dotInfo() + "\n";
            return info;
        }
};
```

Test run

```
int main()
{
    Polygon p1;
    p1.addCornerPoints();
    cout << p1.getCorners();

}
```

Output

```
Corner points:
(41,17)
(34,0)
(19,24)
(28,8)
(12,14)
(5,45)
(31,27)
(11,41)
(45,42)
(27,36)
```

What if we do not know the amount of parts in a container class?

We define a very large fixed array (not good solution)

We can dynamical data structure (e.g. Vector or ArrayList)

Example: Class PointSet contains 1 to n points.

We can now use e.g. vector:

```
vector<Dot> points;
```

Vector has several methods to be used:

- vector::vector
- vector::~vector
- ***member functions:***
  - vector::assign
  - vector::at
  - vector::back
  - vector::begin
  - vector::capacity
  - vector::cbegin
  - vector::cend
  - vector::clear
  - vector::crbegin
  - vector::crend
  - vector::data
  - vector::emplace
  - vector::emplace_back
  - vector::empty
  - vector::end
  - vector::erase
  - vector::front
  - vector::get_allocator
  - vector::insert
  - vector::max_size
  - vector::operator=
  - vector::operator[]
  - vector::pop_back
  - vector::push_back
  - vector::rbegin
  - vector::rend
  - vector::reserve
  - vector::resize
  - vector::shrink_to_fit
  - vector::size
  - vector::swap

Codes

```
class Dot {
    private:
        int x;
        int y;
    public:
        Dot()
        {
        }
        Dot(int xx, int yy)
        {
            x = xx;
            y = yy;
        }
        void setDot(int xx, int yy)
        {
            x = xx;
            y = yy;
        }
        int getX()
        {
            return x;
        }
        int getY()
        {
            return y;
        }
        string dotInfo()
        {
            string info = "("  + to_string(x) + "," + to_string(y) + ")";
            return info;
        }

};
```

```
class Pointset {
    private:
        vector<Dot> points;

    public:
        void addPoint()
        {

            int xx = rand() % 50;
            int yy = rand() % 50;
            Dot d1(xx,yy);
            points.push_back(d1);

        }

        string getPoints()
        {
            string info = "Points: \n";
            for (int k = 0; k < points.size(); k++)
               info += points[k].dotInfo() + "\n";
            return info;
        }
};
```

Test run

```
int main()
{
  Pointset set1;
  set1.addPoint();
  set1.addPoint();
  set1.addPoint();
  set1.addPoint();
  set1.addPoint();

  cout << set1.getPoints();

}
```

Output

```
Points:
(41,17)
(34,0)
(19,24)
(28,8)
(12,14)
```

Try examples!

Note: you can create dynamic data structures by yourself, too.

OR you can try with different storaged that C++ STL offers.

Part 2 END

C++ to all

C++ and classes

Part 3

Table of Contents

Intro

Class is a data type, template that is used to model some abstract or concrete concept.

Object is a variable.

How to model a class? UML

is used.

Unified Modeling Language

UML-tools

StarUML, argoUML, Ms Visio

Inheritance (generalization, specialization)

When 2 or more classes have common (same) features.

Concepts:

Base class, parent class, mother class, general class

Benefits:

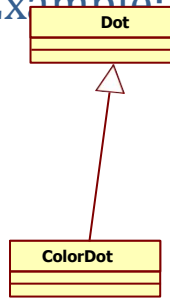
Same code needs not be written several times.

Examples

Person (base class)

Student, teacher, employee (child classes)

Example: Dot -> ColorDot



Codes

```
class Dot {
    private:
        int x;
        int y;
    public:
        Dot()
        {
        }
        Dot(int xx, int yy)
        {
            x = xx;
            y = yy;
        }
        void setDot(int xx, int yy)
        {
            x = xx;
            y = yy;
        }
        int getX()
        {
            return x;
        }
        int getY()
        {
            return y;
        }
        string dotInfo()
        {
            string info = "("  + to_string(x) + "," + to_string(y) + ")";
            return info;
        }

};
```

```
class ColorDot: public Dot
{
    private:
        string color;

    public:
        void setColor(string c)
        {
            color = c;
        }
        string getColor()
        {
            return color;
        }

};
```

Test run

```
int main()
{
    Dot d1;
    d1.setDot(5,6);
    cout << d1.dotInfo() << endl;

    ColorDot cd1;
    cd1.setColor("Black");
    cout << cd1.getColor() << endl;
    cd1.setDot(10,10);
    cout << cd1.dotInfo() << endl;

}
```

Output

```
(5,6)
Black
(10,10)
```

About constructors

Constructor is a special method that is used to create objects.

Default constructor takes no parameters.

Class can have 1 or more constructors that have parameters.

Here is an example that shows when and which constructors are used in inheritance. Dot has these constructors: print statements are just to get info about the usage.

```
public:
    Dot()
    {
        cout << "Dot created!\n";
    }
    Dot(int xx, int yy)
    {
        x = xx;
        y = yy;
        cout << "Dot created!\n";
    }
```

ColorDot has this constructor

```
public:
    ColorDot()
    {
        cout << "Colordot created!\n";
    }
```

We run this code

```
int main()
{
    Dot d1;
    d1.setDot(5,6);
    cout << d1.dotInfo() << endl;

    ColorDot cd1;
    cd1.setColor("Black");
    cout << cd1.getColor() << endl;
    cd1.setDot(10,10);
    cout << cd1.dotInfo() << endl;
}
```

Output

```
Dot created!
(5,6)
Dot created!
Colordot created!
Black
(10,10)
```

Explanation

```
int main()
{
    Dot d1;
    d1.setDot(5,6);
    cout << d1.dotInfo() << endl;

    ColorDot cd1;
    cd1.setColor("Black");
    cout << cd1.getColor() << endl;
    cd1.setDot(10,10);
    cout << cd1.dotInfo() << endl;
}
```

```
Dot created!
(5,6)
Dot created!
Colordot created!
Black
(10,10)
```

SO, ColorDot has 2 parts: Dot and ColorDot.

Constructors and association

Here is an example: triangle that is defined by its cornerpoints.

```
class Triangle
{
    private:
        Dot corner1, corner2, corner3;

    public:
        Triangle()
        {
            cout << "Triangle created!\n";
        }

        void setCorners(Dot a, Dot b, Dot c)
        {
            corner1 = a;
            corner2 = b;
            corner3 = c;
        }
};


int main()
{
    Dot d1;

    Triangle tr1;
}
```

Output

```
Dot created!
Dot created!
Dot created!
Dot created!
Triangle created!
```

Note:

Objects are created automatically when a new triangle object is created.

```
class Triangle
{
    private:
        Dot corner1, corner2, corner3;

    public:
        Triangle()
        {
            cout << "Triangle created!\n";
        }

        void setCorners(Dot a, Dot b, Dot c)
        {
            corner1 = a;
            corner2 = b;
            corner3 = c;
        }
};


int main()
{
    Dot d1;

    Triangle tr1;
}
```

These attributes (member variables) are not references by default in C++.

But in Java and C# they are references:
- you have to create objects and put references to refer to them

In C++, in this example, cornerpoints are created when a triangle is created.

You can avoid this by using pointers!

Pointer attributes used

```
class Triangle
{
    private:
        Dot *corner1, *corner2, *corner3;

    public:
        Triangle()
        {
            cout << "Triangle created!\n";
        }

        void setCorners(Dot *a, Dot *b, Dot *c)
        {
            corner1 = a;
            corner2 = b;
            corner3 = c;
        }
};


int main()
{
    Dot d1;

    Triangle tr1;
}
```

```
Dot created!
Triangle created!
```

Triangle corners have not created yet…

Let's create the triangle with corners, too:

```
int main()
{
    Dot d1(2,3);
    Dot d2(5,7);
    Dot d3(5,1);

    Triangle tr1;
    tr1.setCorners(&d1, &d2, &d3);

}
```

```
Dot created!
Dot created!
Dot created!
Triangle created!
```

Shortly about copy constructor

When a copy of an object is created, attribute values have to be copied to object. If attributes are pointers, new memory has to be allocated first.

As a patameter type reference is commonly used in copy constructor. Normally the fact that the object that is to be copied has not to be chanced, is defined by using const keyword.

Code

```
Dot(const Dot & sourceDot)
{
    x = sourceDot.x;
    y = sourceDot.y;
    cout << "Dot is copied!\n";
}
```

Test run

```
int main()
{
    Dot d1(2,3);
    cout << d1.dotInfo() << endl;

    // make a copy of d1
    Dot d2(d1);
    cout << d2.dotInfo() << endl;
}
```

Output

```
Dot created!
(2,3)
Dot is copied!
(2,3)
```

When (as said before) we have pointers as attributes and we need to allocate memory for values. we have to be more carefull so that we do not share the same memory.

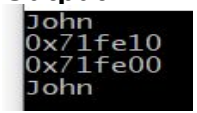
Example: pointers

```
class Person {
    private:
    char * name;
    public:

    Person() {
    }

    Person(char *  n) {
       int size = strlen(n);
       name = new char[size];
       strcpy(name, n);
    }

    Person(const Person & person) {
       name = person.name;
    }

    void setName(char * n) {
       int size = strlen(n);
       name = new char[size];
       strcpy(name, n);
    }

    char * getName() {
       return name;
    }

};
```

Now we have person's name as a char array (pointer is used).

So, when we add value to name, we have to allocate memory.

Output



Here we have an example that shows how program crashes when memory has not been allocated inside the copy constructor:

```
class Person {
    private:
    int * id;

    public:
    Person() {
    }

    Person(int v) {
       id = new int(v);
    }

    Person(const Person & person) {
        *id = *person.id;
    }

    void setId(int v) {
       id = new int(v);
    }


    int *  getIdAd() {
       return id;
    }

    int  getId() {
       return *id;
    }
};
```

Test run

```
int main()
{
    Person p1(10);
    cout << p1.getId() << endl;
    cout << p1.getIdAd() << endl;

    // copying
    Person p2 = p1;
    cout << p2.getId() << endl;
    cout << p2.getIdAd() << endl;

    Person p3 = p2;
    cout << p3.getId() << endl;
    cout << p3.getIdAd() << endl;
}
```

Output

```
10
0x191510
10
0x191390

--------------------------------
Process exited after 1.989 seconds with return value 3221225477
Press any key to continue . . .
```

Updating the copy constructor

```
Person(const Person & person) {
    id = new int();
    *id = *person.id;
}
```

Output

```
10
0xcf1510
10
0xcf1530
10
0xcf1550

--------------------------------
Process exited after 0.7251 seconds with return value 0
Press any key to continue . . .
```

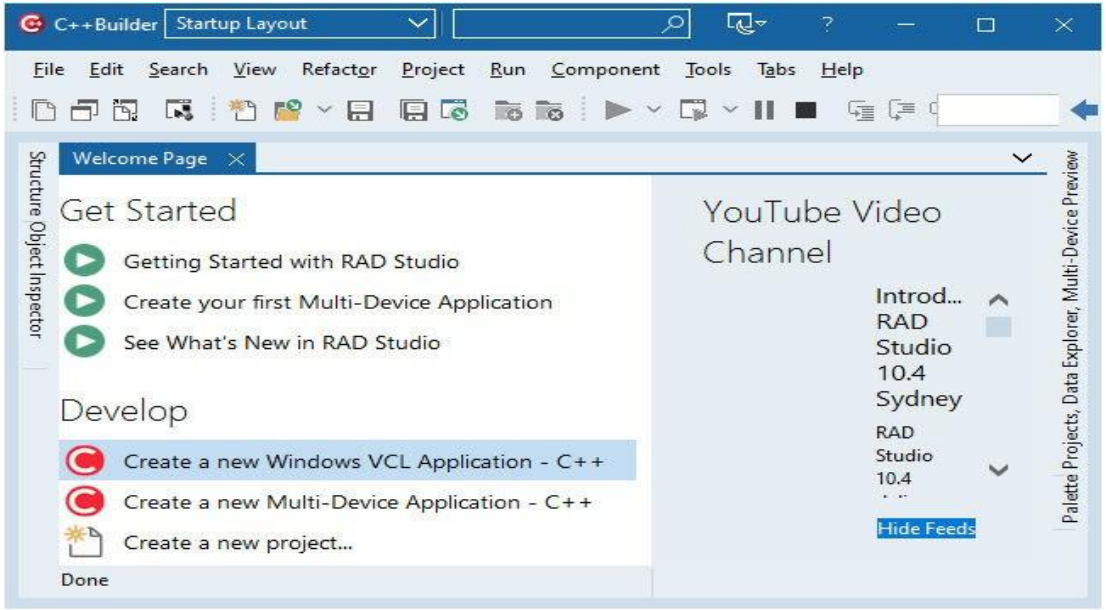
Exercise

Create a Person class that is the base class for a Student class.
Person has a name.

Student has a student code.

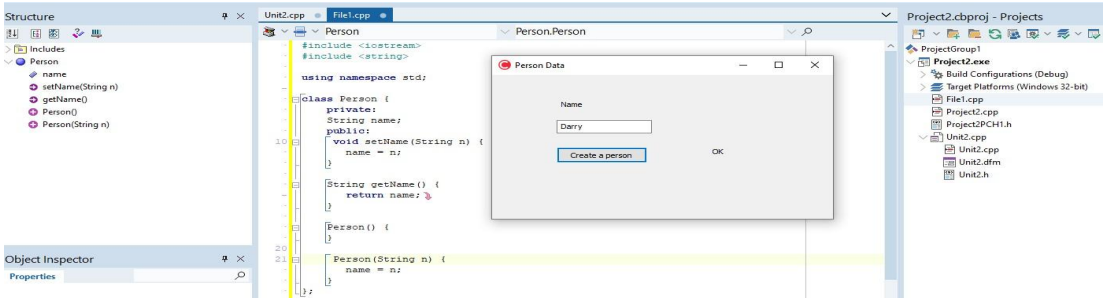
Then create a class named Group that contains 1 – n students.
Add some students and print out information about students.

Short look at the GUI tool



Example: Gui

Person class code is taken to C++Builder project.



We go on studying GUI more later.

Part 3 End

Thank You!

C++ to all

C++ and classes

Part 4

Table of Contents

Intro

Class is a data type, template that is used to model some abstract or concrete concept.

Object is a variable.

How to model a class? UML is used.

Unified Modeling Language

UML-tools

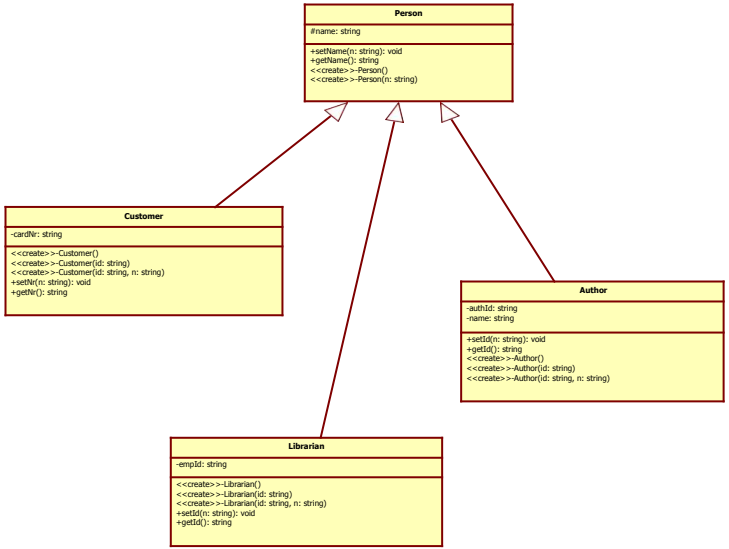
StarUML, argoUML, Ms Visio

Application Prototype: Several classes

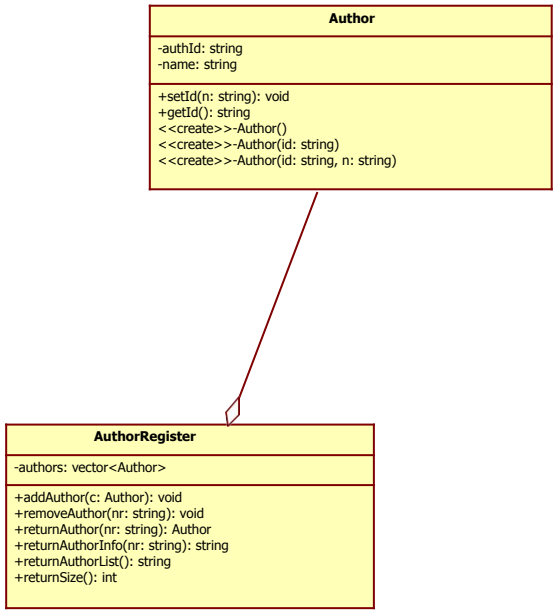
Library demo

Class diagrams

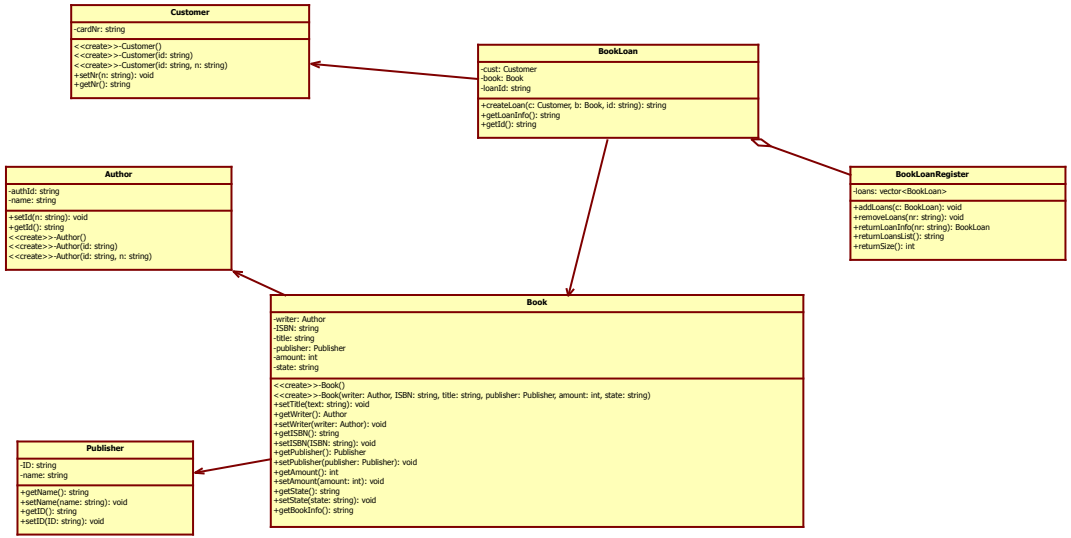
Part A



Part B

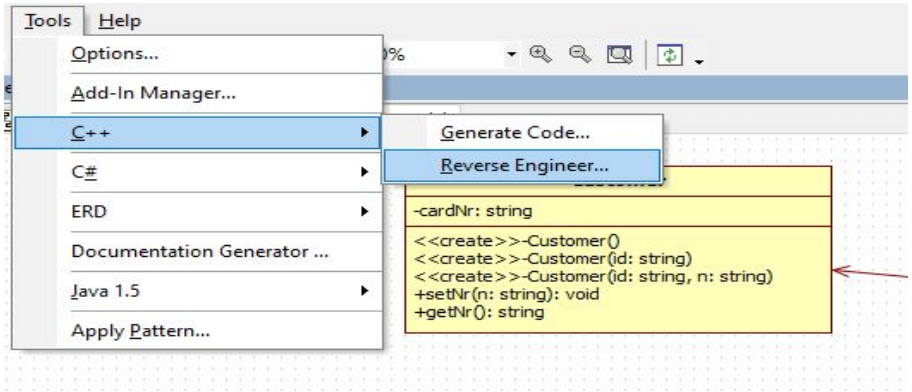

Author
-authId: string
-name: string
+setId(n: string): void
+getId(): string
<<create>>-Author()
<<create>>-Author(id: string)
<<create>>-Author(id: string, n: string)
AuthorRegister
-authors: vector<Author>
+addAuthor(c: Author): void
+removeAuthor(nr: string): void
+returnAuthor(nr: string): Author
+returnAuthorInfo(nr: string): string
+returnAuthorList(): string
+returnSize(): int

Part C


Customer
-cardNr: string
<<create>>-Customer()
<<create>>-Customer(id: string)
<<create>>-Customer(id: string, n: string)
+setNr(n: string): void
+getNr(): string
BookLoan
-cust: Customer
-book: Book
-loanId: string
+createLoan(c: Customer, b: Book, id: string): string
+getLoanInfo(): string
+getId(): string
Author
-authId: string
-name: string
+setId(n: string): void
+getId(): string
<<create>>-Author()
<<create>>-Author(id: string)
<<create>>-Author(id: string, n: string)
BookLoanRegister
-loans: vector<BookLoan>
+addLoans(c: BookLoan): void
+removeLoans(nr: string): void
+returnLoanInfo(nr: string): BookLoan
+returnLoansList(): string
+returnSize(): int
Book
-writer: Author
-ISBN: string
-title: string
-publisher: Publisher
-amount: int
-state: string
<<create>>-Book()
<<create>>-Book(writer: Author, ISBN: string, title: string, publisher: Publisher, amount: int, state: string)
+setTitle(text: string): void
+getWriter(): Author
+setWriter(writer: Author): void
+getISBN(): string
+setISBN(ISBN: string): void
+getPublisher(): Publisher
+setPublisher(publisher: Publisher): void
+getAmount(): int
+setAmount(amount: int): void
+getState(): string
+setState(state: string): void
+getBookInfo(): string
Publisher
-ID: string
-name: string
+getName(): string
+setName(name: string): void
+getID(): string
+setID(ID: string): void

UML diagram: generated from code

In this case class diagram was generated from the C++ code:



Here are C++ codes:

```
class Person {
      protected:
       string name;

      public:
         void setName(string n)
         {
             name = n;
         }

         string getName()
         {
             return name;
         }

          Person()
         {

         }

         Person(string n)
         {
             name = n;
         }
};
```

```
class Customer: public Person {
    private:
        string cardNr;

    public:
        Customer()
            { }

        Customer(string id)
            {
                cardNr = id;
            }
        Customer(string id, string n)
            {
               cardNr = id;
                name = n;
            }


        void setNr(string n)
            {
                cardNr = n;
            }

        string getNr()
            {
            return cardNr;
            }

};
```

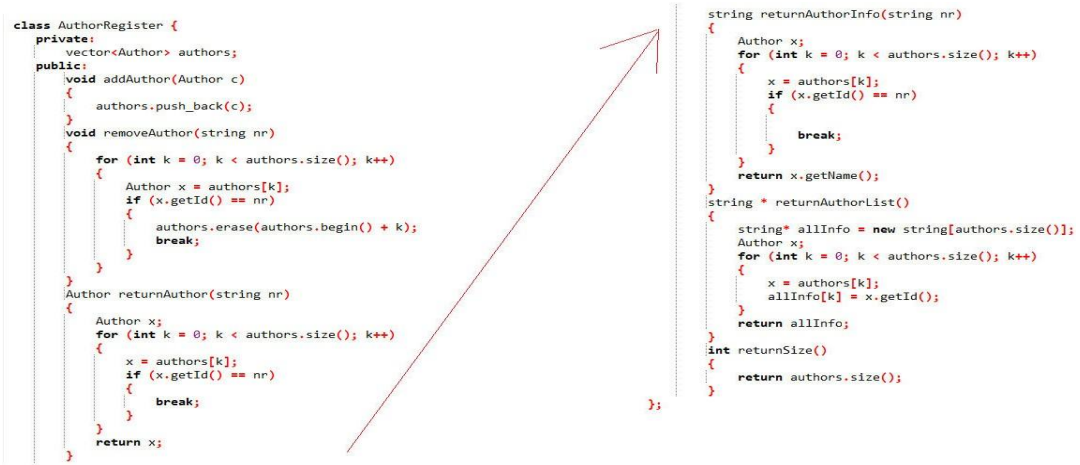
```
class Librarian: public Person{
    private:
        string empId;

    public:

        Librarian()
        { }

        Librarian(string id)
        {
            empId = id;
        }
        Librarian(string id, string n)
        {
            empId = id;
            name = n;
        }
         void setId(string n)
        {
            empId = n;
        }

        string getId()
        {
            return empId;
        }
};
```

```
class Author: public Person {

    private:
        string authId;
        string name;

    public:
        void setId(string n)
        {
            authId = n;
        }

        string getId()
        {
            return authId;
        }

        Author()
        { }

        Author(string id)
        {
            authId = id;
        }
        Author(string id, string n)
        {
            authId = id;
            name = n;
        }

};
```

```
class Publisher {
    private:
        string ID;
        string name;

    public:
        string getName() {
        return name;
    }

        void setName(string name) {
           this->name = name;
        }
         string getID() {
            return ID;
        }

        void setID(string ID) {
            this->ID = ID;
        }

};
```

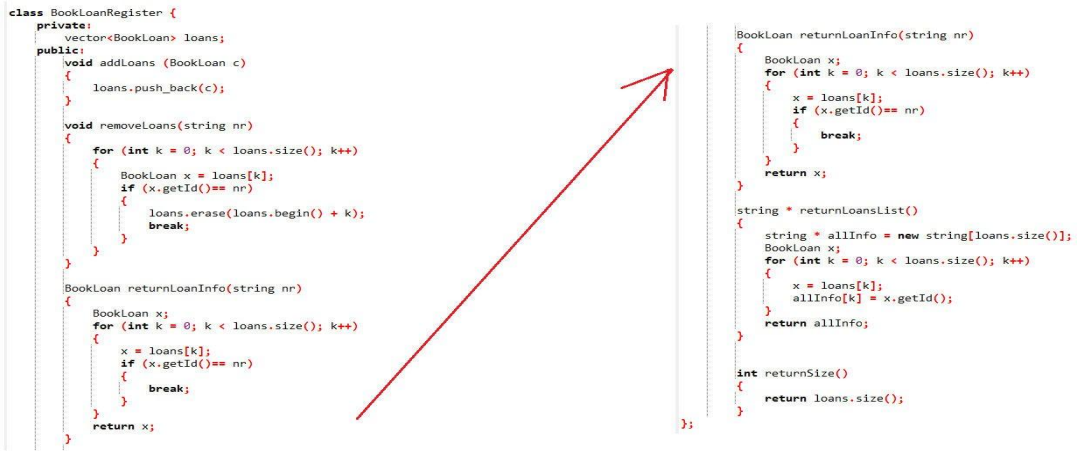
```
class Book {
    private:
        Author writer;
        string ISBN;
        string title;
        Publisher publisher;
        int amount;
        string state;
    public:
        Book() {          }
        Book(Author writer, string ISBN, string title, Publisher publisher, int amount, string state) {
            this->writer = writer;
            this->ISBN = ISBN;
            this->title = title;
            this->publisher = publisher;
            this->amount = amount;
            this->state = state;
        }
        void setTitle(string text) {  this->title = text; }
        Author getWriter() { return writer;  }
        void setWriter(Author writer) { this->writer = writer;}
        string getISBN() { return ISBN;}
        void setISBN(string ISBN) { this->ISBN = ISBN; }
        Publisher getPublisher() { return publisher; }
        void setPublisher(Publisher publisher) { this->publisher = publisher; }
        int getAmount() { return amount; }
        void setAmount(int amount) { this->amount = amount; }
        string  getState() { return state; }
        void setState(string state) { this->state = state; }
        string getBookInfo()
        {
            string info = "Book's name is " + title + "\n";
            info += "books state is " + state + "\n";
            info += "amount of that book is " + to_string(amount) + "\n";
            return info;
        }
};
```

```
class BookLoan {
        private:
         Customer cust;
         Book book;
         string loanId;
        public:
            string createLoan(Customer c, Book b, string id)
            {
                cust = c;
                book = b;
                loanId = id;
                return "loan done!";
            }
            string getLoanInfo()
            {
                string info = "Customer is " + cust.getName();
                info += " and book is " + book.getISBN();
                info += " and loanid is " + loanId;

                return info;

            }

             string getId()
            {
                return loanId;
            }
};
```



```
class AuthorRegister {
    private:
        vector<Author> authors;
    public:
        void addAuthor(Author c)
        {
            authors.push_back(c);
        }
        void removeAuthor(string nr)
        {
            for (int k = 0; k < authors.size(); k++)
            {
                Author x = authors[k];
                if (x.getId() == nr)
                {
                    authors.erase(authors.begin() + k);
                    break;
                }
            }
        }
        Author returnAuthor(string nr)
        {
            Author x;
            for (int k = 0; k < authors.size(); k++)
            {
                x = authors[k];
                if (x.getId() == nr)
                {
                    break;
                }
            }
            return x;
        }
};
```

```
string returnAuthorInfo(string nr)
{
    Author x;
    for (int k = 0; k < authors.size(); k++)
    {
        x = authors[k];
        if (x.getId() == nr)
        {
            break;
        }
    }
    return x.getName();
}
string * returnAuthorList()
{
    string* allInfo = new string[authors.size()];
    Author x;
    for (int k = 0; k < authors.size(); k++)
    {
        x = authors[k];
        allInfo[k] = x.getId();
    }
    return allInfo;
}
int returnSize()
{
    return authors.size();
}
```



```
class BookLoanRegister {
    private:
        vector<BookLoan> loans;
    public:
        void addLoans (BookLoan c)
        {
            loans.push_back(c);
        }

        void removeLoans(string nr)
        {
            for (int k = 0; k < loans.size(); k++)
            {
                BookLoan x = loans[k];
                if (x.getId()== nr)
                {
                    loans.erase(loans.begin() + k);
                    break;
                }
            }
        }

        BookLoan returnLoanInfo(string nr)
        {
            BookLoan x;
            for (int k = 0; k < loans.size(); k++)
            {
                x = loans[k];
                if (x.getId()== nr)
                {
                    break;
                }
            }
            return x;
        }

        BookLoan returnLoanInfo(string nr)
        {
            BookLoan x;
            for (int k = 0; k < loans.size(); k++)
            {
                x = loans[k];
                if (x.getId()== nr)
                {
                    break;
                }
            }
            return x;
        }

        string * returnLoansList()
        {
            string * allInfo = new string[loans.size()];
            BookLoan x;
            for (int k = 0; k < loans.size(); k++)
            {
                x = loans[k];
                allInfo[k] = x.getId();
            }
            return allInfo;
        }

        int returnSize()
        {
            return loans.size();
        }
};
```

Test run

```
int main()
{
    Author a1("123");
    a1.setName("Darry");

    AuthorRegister ar;
    ar.addAuthor(a1);

    cout << ar.returnAuthorInfo("123");

    Publisher publisher1;
    publisher1.setName("OTAVA");

    Book book1;
    book1.setAmount(10);
    book1.setISBN("ABC");
    book1.setPublisher(publisher1);
    book1.setTitle("Horses go");
    book1.setWriter(a1);

    book1.setState("borrowable");

    cout << book1.getBookInfo();

    Customer customer1("01","Molly");

    BookLoan loan1;
    cout << loan1.createLoan(customer1, book1, "Loan2020/1");

    cout << endl;

    cout << loan1.getLoanInfo();


}
```

Output

```
DarryBook's name is Horses go
books state is borrowable
amount of that book is 10
loan done!
Customer is Molly and book is ABC and loanid is Loan2020/1
--------------------------------
```

Try it!

We are going to create a GUI based library proto in the next part!

Thank You!

C++ to all

C++ and classes

This document continues with Library demo

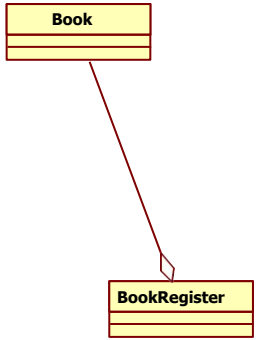
Table of Contents

New Library Demo features are

- Book Register
- Publisher Register
- Customer Register
- Librarian Register

Book Register

As an example here shows Book Register diagram and code:



```
class Book {
    private:
        Author writer;
        string ISBN;
        string title;
        Publisher publisher;
        int amount;
        string state;
    public:
        Book() {          }
        Book(Author writer, string ISBN, string title, Publisher publisher, int amount, string state) {
            this->writer = writer;
            this->ISBN = ISBN;
            this->title = title;
            this->publisher = publisher;
            this->amount = amount;
            this->state = state;
        }
        void setTitle(string text) {  this->title = text; }
        string getTitle() {  return title; }
        Author getWriter() { return writer;  }
        void setWriter(Author writer) { this->writer = writer;}
        string getISBN() { return ISBN;}
        void setISBN(string ISBN) { this->ISBN = ISBN; }
        Publisher getPublisher() { return publisher; }
        void setPublisher(Publisher publisher) { this->publisher = publisher; }
        int getAmount() { return amount; }
        void setAmount(int amount) { this->amount = amount; }
        string  getState() { return state; }
        void setState(string state) { this->state = state; }
        string getBookInfo()
        {
            string info = "Book's name is " + title + "\n";
            info += "books state is " + state + "\n";
            info += "amount of that book is " + to_string(amount) + "\n";
            return info;
        }
    };
```

```
class BookRegister {
    private:
        vector <Book> books;
    public:
        void addBook(Book c)
        {
            books.push_back(c);
            saveBookDataToDB(c);
        }
       void saveBookDataToDB(Book c)
        {
            ofstream f2;
            f2.open("books.txt", ios::app);
            f2 << c.getISBN() << "\n";
            f2 << c.getTitle() << "\n";
            f2.close();
        }
        void removeBook(string isbn)
        {
            for (int k = 0; k < books.size(); k++)
            {
                Book x = books[k];
                if (x.getISBN() == isbn)
                {
                    books.erase(books.begin() + k);
                    break;
                }
            }
        }
        Book returnBook(string isbn)
        {
            Book x;
            for (int k = 0; k < books.size(); k++)
            {
                x = books[k];
                if (x.getISBN() == isbn)
                {
                    break;
                }
            }
            return x;
        }
        string returnBookInfo(string isbn)
        {
            Book x;
            for (int k = 0; k < books.size(); k++)
            {
                x = books[k];
                if (x.getISBN() == isbn)
                {
                    break;
                }
            }
            return x.getBookInfo();
        }
        string * returnBookList()
        {
            string * allInfo = new string[books.size()];
            Book x;
            for (int k = 0; k < books.size(); k++)
            {
                x = books[k];
                allInfo[k] += x.getTitle();
            }
            return allInfo;
        }
        int returnSize()
        {
            return books.size();
        }
};
```

Using textfiles

New feature is also saving to a file.

Take a look at these methods.

```
public:
    void addBook(Book c)
    {
        books.push_back(c);
        saveBookDataToDB(c);
    }
   void saveBookDataToDB(Book c)
    {
        ofstream f2;
        f2.open("books.txt", ios::app);
        f2 << c.getISBN() << "\n";
        f2 << c.getTitle() << "\n";
        f2.close();
    }
```

Method void saveBookDataToDB(Book c) saves book data to a file, called "books.txt".

This is just an example but really on way to use permanent storages to data. You san write info to textfiles and read it.

Normally we of course use databases but this is a light and easy way to be used is specific situations.

Test run now

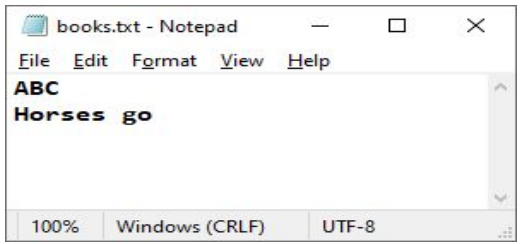
```
int main()
{
    Author a1("123");
    a1.setName("Darry");
    AuthorRegister ar;
    ar.addAuthor(a1);

    Publisher publisher1;
    publisher1.setName("OTAVA");
    publisher1.setID("Pub_1");

    Book book1;
    book1.setAmount(10);
    book1.setISBN("ABC");
    book1.setPublisher(publisher1);
    book1.setTitle("Horses go");
    book1.setWriter(a1);
    book1.setState("borrowable");

    Customer customer1("01","Molly");

    BookLoan loan1;
    cout << loan1.createLoan(customer1, book1, "Loan2020/1");
    cout << endl;

    BookRegister br;
    br.addBook(book1);
    cout << endl;
    cout << br.returnBookInfo("ABC");
```

Output

```
loan done!

Book's name is Horses go
books state is borrowable
amount of that book is 10
```

What about the text file?

It has been created and book info is saved there:



Try it!

We are going to create a GUI based library proto in the next part!

Thank You!

C++ to all

C++ and classes

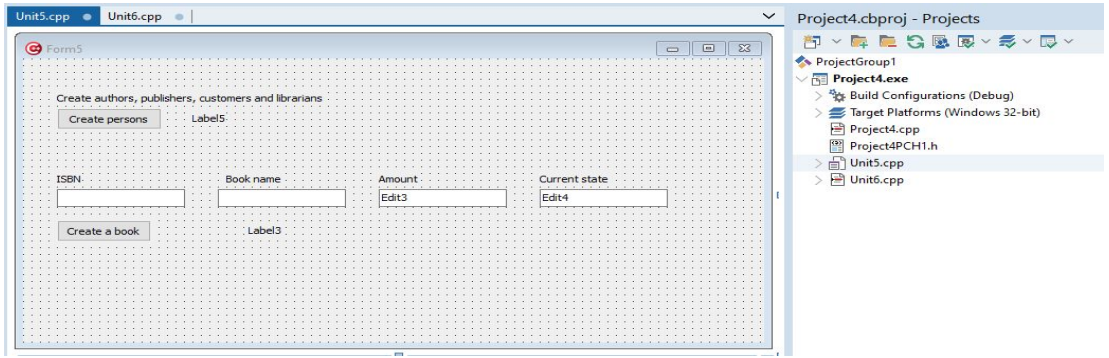
Part 6

Table of Contents

Gui

We create a GUI prototype with C++Builder.

Here is the new project.

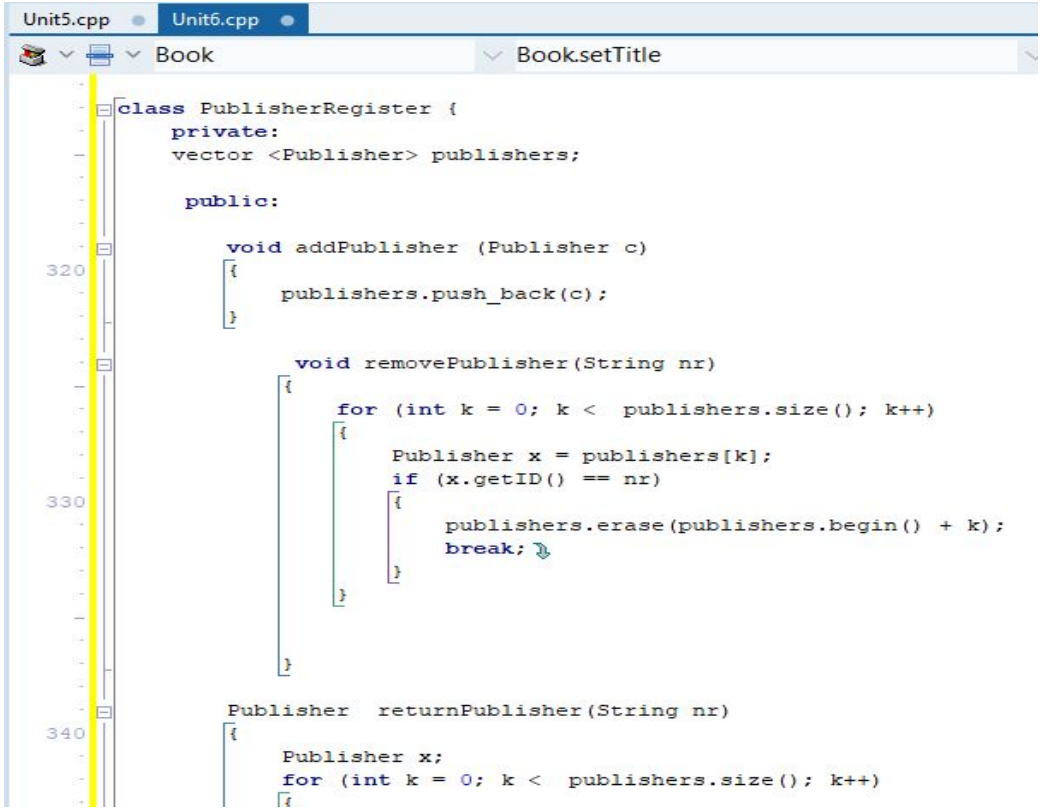


GUI code

Gui code is on Unit5.cpp.

ALl class codes have been added to Unit6.cpp.

Here is a sample of that file:

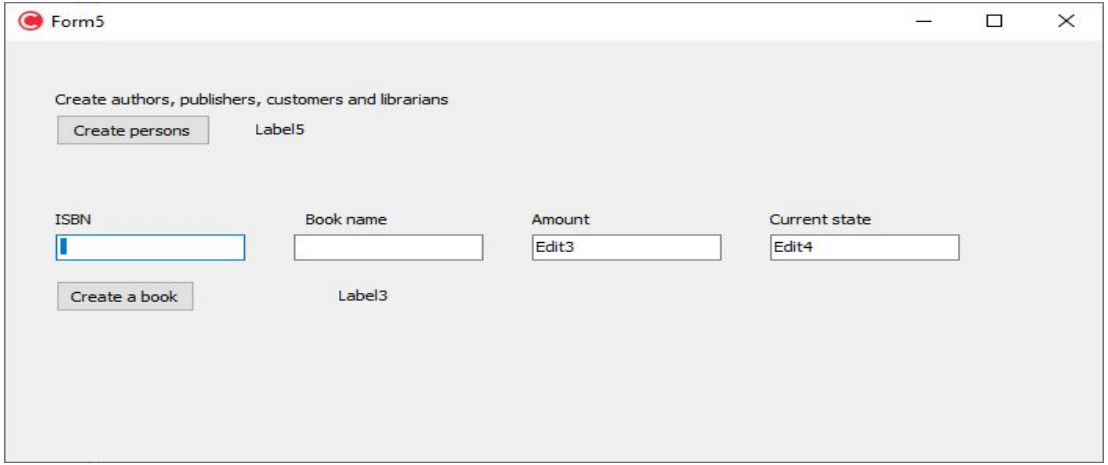

Unit5.cpp
Unit6.cpp
Book
Book.setTitle
class PublisherRegister {
private:
vector <Publisher> publishers;
public:
void addPublisher (Publisher c)
{
publishers.push_back(c);
}
void removePublisher(String nr)
{
for (int k = 0; k < publishers.size(); k++)
{
Publisher x = publishers[k];
if (x.getID() == nr)
{
publishers.erase(publishers.begin() + k);
break;
}
}
}
Publisher returnPublisher(String nr)
{
Publisher x;
for (int k = 0; k < publishers.size(); k++)
320
330
340

Main difference to standard C++ code is String: C++Builder has String type with capital S. Standard C++ has name string (small letters only).

This is a demo: we just create all person objects and add then to registers. Then we create a new book and print info.

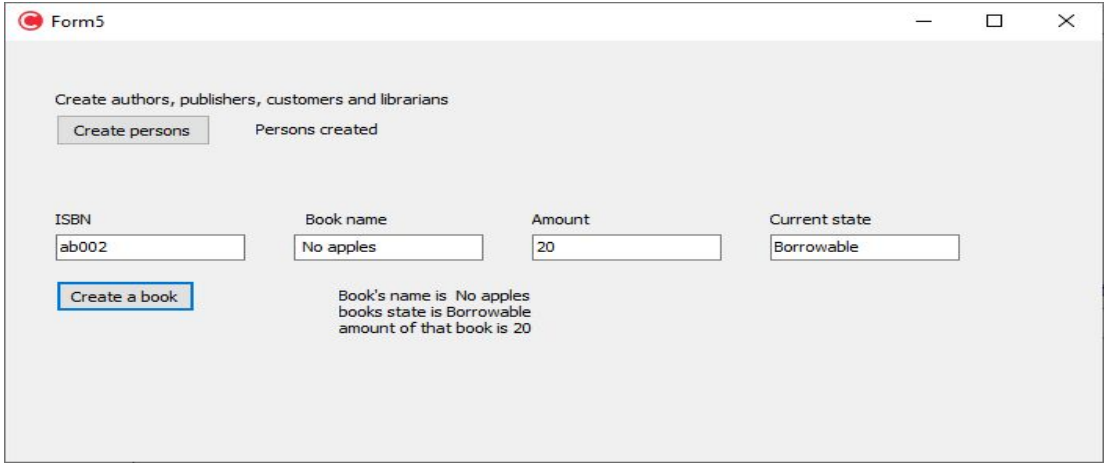
Test run

Let's run it:



With button "Create persons" we create all persons..

With button "Create a book" we add a new book to the register.


Form5
Create authors, publishers, customers and librarians
Create persons
Persons created
ISBN
ab002
Book name
No apples
Amount
20
Current state
Borrowable
Create a book
Book's name is No apples
books state is Borrowable
amount of that book is 20

What to add?

Add new features now. There are a lot of choices:

- e.g. we could use list controls to show contents of all registers
- add book loans (user can choose the book from the list)
- add searching methods
- add exception handling
- try tabbed form (you have easily too much info on a single screen)
- and so on

Then of course we could use text files to store information: then when we run the program we can read basis info from textfiles.

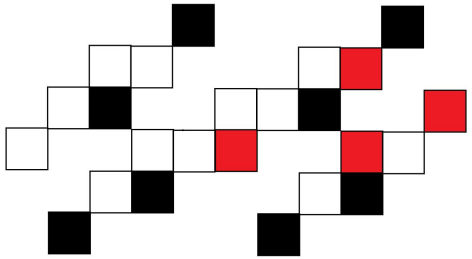
Only new items need more job (inputting data) - (we avoid working extra when coding but testing really needs some contents...).

Try it!
Make it more versatile...

Thank You!

Kakelino’s Code School

SOME ICT IDEAS

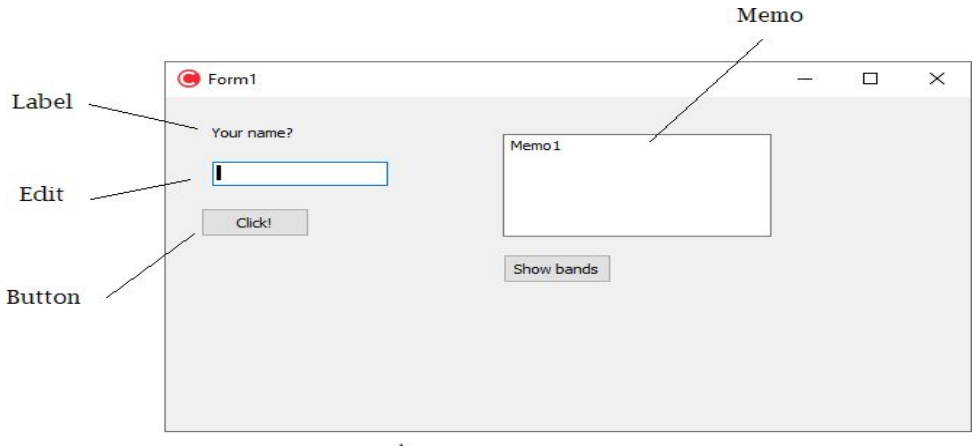


C++Builder - Presentation

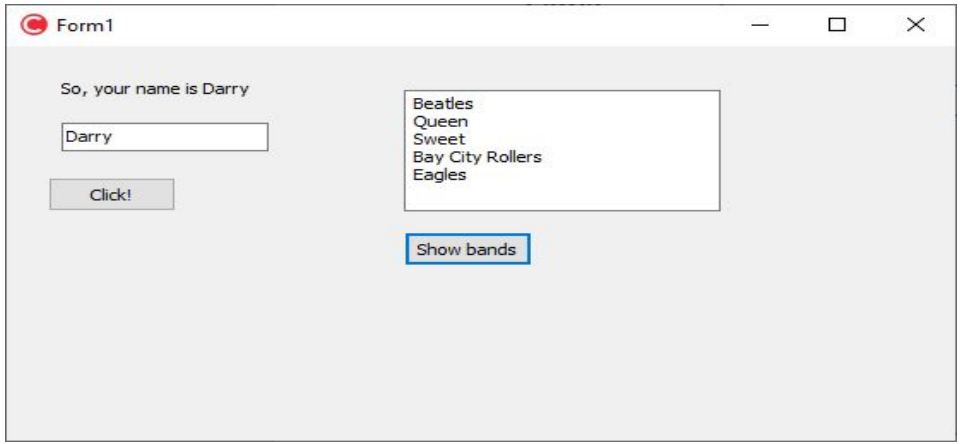
Table of Contents

Basic controls

Start a new project. Add there some basic controls:



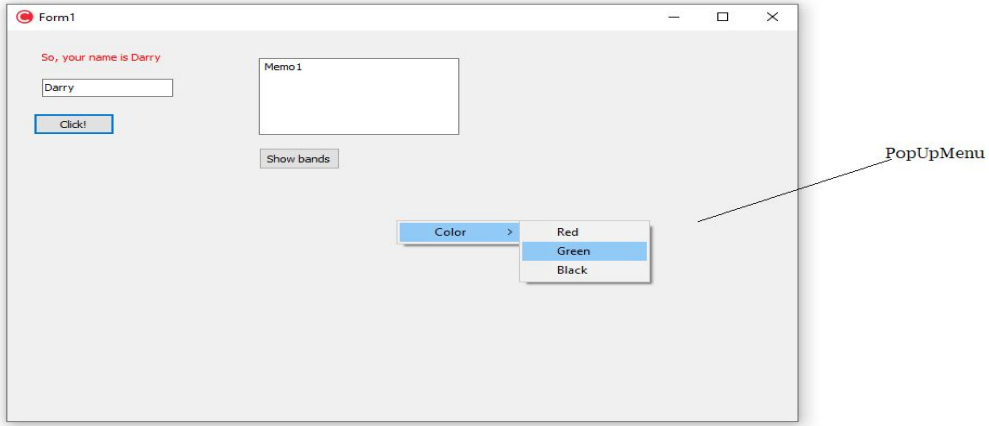
Test run


Form1
So, your name is Darry
Darry
Click!
Beatles
Queen
Sweet
Bay City Rollers
Eagles
Show bands

Codes:

```
String bands[] =
{
   "Beatles",
   "Queen",
   "Sweet",
   "Bay City Rollers",
   "Eagles"
};
//---------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
   String a = Edit1->Text;
   Label1->Caption = "So, your name is " +  a;
}
//---------------------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
    Memo1->Clear();
    int s = sizeof(bands)/sizeof(bands[0]);
    for (int k = 0; k < s; k++)
    {
        Memo1->Lines->Append(bands[k]) ;
    }
```

PopUpMenu



Popupmenu has been added to the form. 3

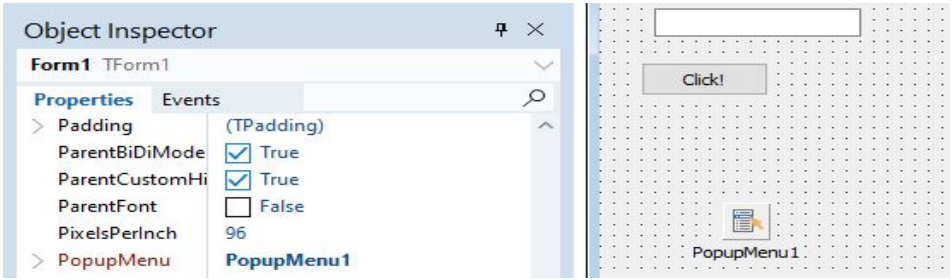
submenus are created.

Menu is connected to the forim itself: when user rightclicks the form, menu opens.

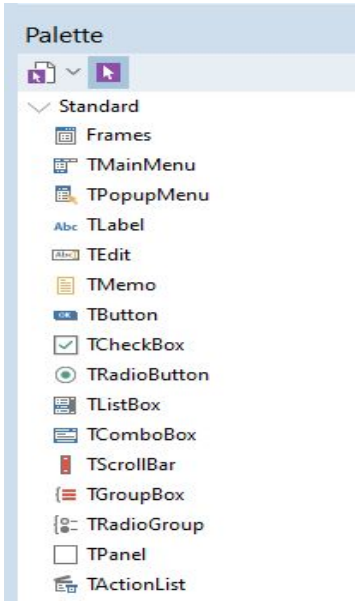
Codes:

```
void __fastcall TForm1::Red3Click(TObject *Sender)
{
    Label1->Font->Color = clRed;
}
//---------------------------------   ----------------------------

void __fastcall TForm1::Red4Click(TObject *Sender)
{
    Label1->Font->Color  = clGreen;
}
//---------------------------------------------------------------

void __fastcall TForm1::Black2Click(TObject *Sender)
{
    Label1->Font->Color  = clBlack;
}
//---------------------------------------------------------------
```

Connection to the form



In standard controls group there are these components



Some controls from other groups: Image, Timer

Image

Timer

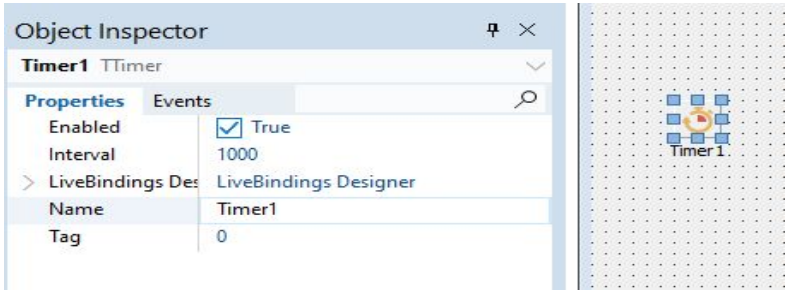
Picture

Add Picture component from Additional group.

And Timer from System group.

Timer

Timer settings

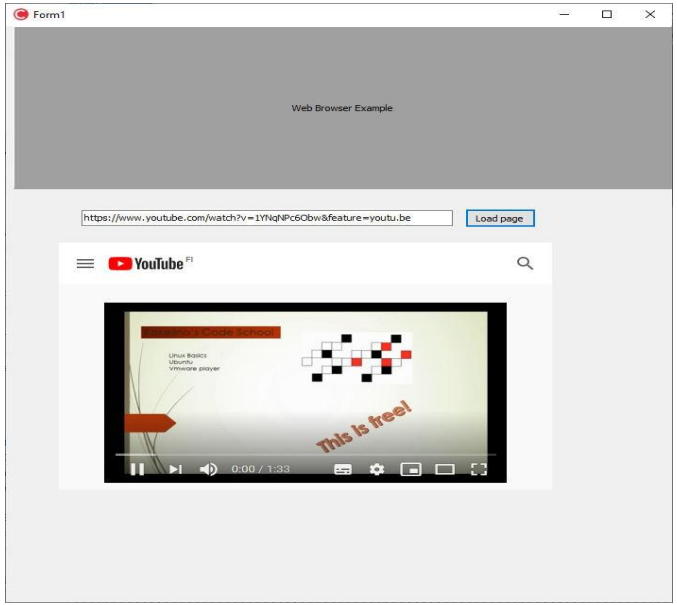


Codes

```
int step = -10;
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
    Image1->Height += step;
    Image1->Width += step;

    if (Image1->Width <= 100)
       step = 10;
    if (Image1->Width >= 300)
       step = -10;

}
```

Web Browser

Let's try Web Browser



Add the component.

Create the code:

```
void __fastcall TForm1::Button3Click(TObject *Sender)
{
    String url = Edit2->Text;

    WebBrowser1->Navigate(url)        ;
}
```

Good!

Try it!