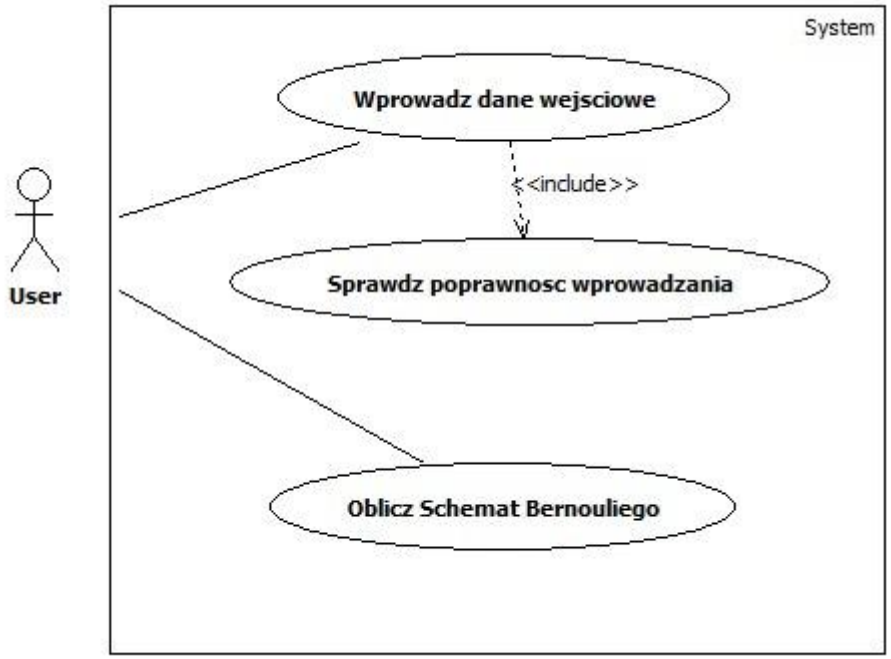
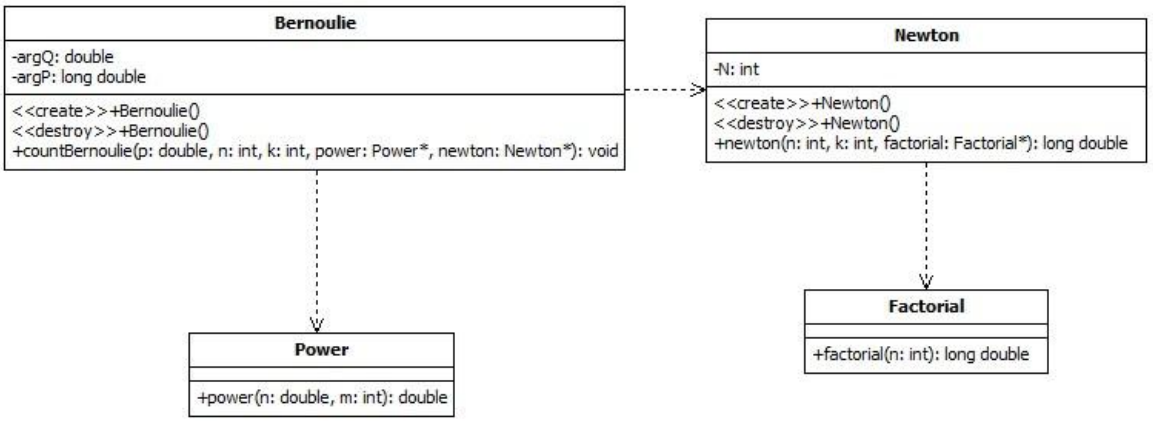
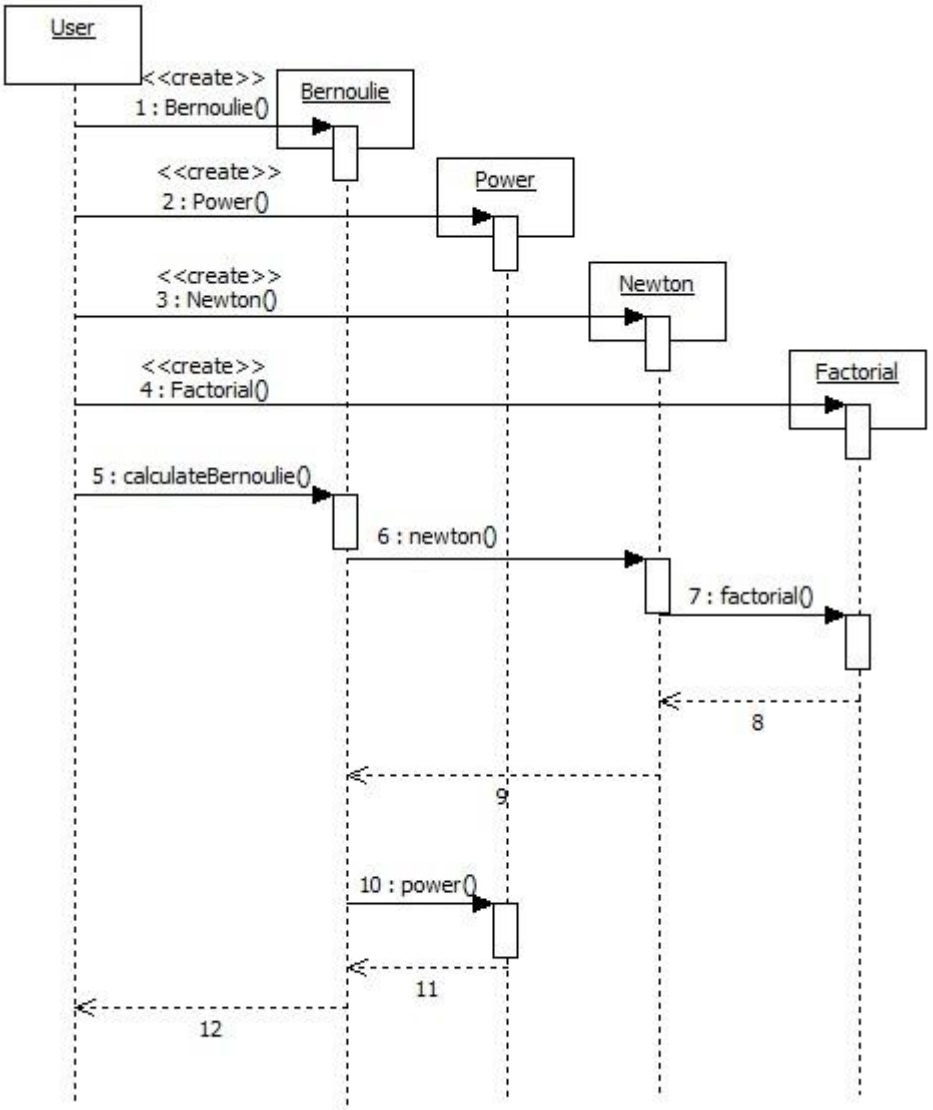
1. Diagram przypadków użycia



2. Model logiczny problemu



3. Diagram sekwencji



4. Implementacja systemu

```
#if !defined(_POWER_H)
#define _POWER_H


class Power {
public:
    double power(double n, int m);
};

#endif  //_POWER_H
```

```
#include "Power.h"

double Power::power(double n, int m) {
    double result = 1;
    for (int i = 0; i < m; i++){
        result = result * n;
    }
    return result;
}
```

```
#if !defined(_FACTORIAL_H)
#define _FACTORIAL_H


class Factorial {
public:
    long double factorial(int n);
};

#endif  //_FACTORIAL_H
```

```
#include "Factorial.h"

long double Factorial::factorial(int n) {
    long double result = 1.0;
    for(int i=1; i<=n; i++){
        result *=i;
    }
    return result;
}
```

```h
#if !defined(_NEWTON_H)
#define _NEWTON_H
#include "Factorial.h"

class Newton {
public:
    Newton();
    ~Newton();
    long double newton(int n, int k, Factorial* factorial);
private:
    int N;
};

#endif  //_NEWTON_H
```

```cpp
#include "Newton.h"

Newton::Newton() {

}

Newton::~Newton() {

}

long double Newton::newton(int n, int k, Factorial* factorial) {
    N = n - k;
    long NbyK = 1;
    if(k >= N) {
        for (int i = k + 1; i <= n; i++) {
            NbyK *= i;
        }
        return (NbyK/factorial->factorial( n: n-k));
    }
    else {
        for(int i = N+1; i <= n; i++) {
            NbyK *= i;
        }
        return (NbyK / factorial->factorial( n: k));
    }
}
```

```h
#if !defined(_BERNOULIE_H)
#define _BERNOULIE_H
#include "Power.h"
#include "Newton.h"
#include <iostream>

using namespace std;

class Newton;
class Bernoulie {
public:
    Bernoulie();
    ~Bernoulie();
    void countBernoulie(double p, int n, int k, Power* power, Newton* newton, Factorial* factorial);
private:
    double argQ;
    long double argP;
};

#endif  //_BERNOULIE_H
```

```cpp
#include "Bernoulie.h"

Bernoulie::Bernoulie() {

}

Bernoulie::~Bernoulie() {

}

void Bernoulie::countBernoulie(double p, int n, int k, Power* power, Newton* newton, Factorial* factorial) {
    argQ = 1 - p;
    auto result : long double = (newton->newton(n,k, factorial) *
                  power->power( n: p, m: k) *
                  power->power( n: argQ, m: n-k));
    cout << "Prawdopodobienstwo, ze w " << n << " doswiadczeniach "
         << k << " razy otrzymamy sukces wynosi " << result <<endl;

}
```

```cpp
#include "Bernoulie.h"

int main() {
    double p;
    int n, k;
    cout << " Podaj p: ";
    cin >> p;
    cout << " Podaj n: ";
    cin >> n;
    cout << " Podaj k: ";
    cin >> k;
    Bernoulie *bernoulie = new Bernoulie();
    Power *power = new Power();
    Newton *newton = new Newton();
    Factorial *factorial = new Factorial();
    bernoulie->countBernoulie(p,n,k,power,newton,factorial);
    delete bernoulie;
    delete power;
    delete newton;
    delete factorial;
    return 0;
}
```

5. Test systemu

```
Podaj p: 0.5
Podaj n: 6
Podaj k: 2
Prawdopodobienstwo, ze w 6 doswiadczeniach 2 razy otrzymamy sukces wynosi 0.234375
```