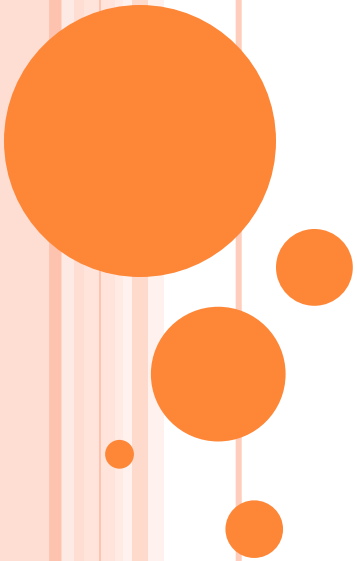


Data Structures and Algorithms

Lab 3

Iuliana Marin

C++ AND OOP

Structure a C++ program as follows:

- Detect the set of tasks and subtasks .
- Write functions for all the tasks .
- Test them.

- 3 key concepts in OOP:
  - Encapsulation – mix data and functions into a single class. Data is not accessed directly. Makes data hiding possible.
  - Inheritance – reuse code between related types.
  - Polymorphism – allow an object to be one of the several types which exist. Determine at runtime which functions are called based on the object's type.

Encapsulation

- Is done through classes.
- In C++ exist public and private access specifiers.
- The things defined inside the class are considered internal details.

Inheritance

- Allows the definition of hierarchies.
- For the example where we have the Animal class, we can define the Dog and Cat as subtypes.

```
public :
    Animal ( const string & myName, const int myAge ): name ( myName), age ( myAge ) {}
    const string getDesc () const
    {
        return name + " with the age " + std::to_string( age );
    }
    const string & getName () const
    {
        return name;
    }
    const int getAge () const
    {
        return age ;
    }
};
```

INHERITANCE

- It is wanted to specify that Dog will inherit the Animal code, but with some extra information.

```
public :
    Dog( const string & myName, const int myAge, const string & myBreedName )
        : Animal ( myName, myAge ), breedName ( myBreedName ) {}
    const string & getBreedName ()
    {
        return breedName ;
    }
    const string getDesc () // Overriding this member function
    {
        return std::to_string(age) + ' ' + name + ": " + breedName;
    }
};

int main()
{
    Dog d("Rolly", 5, "Golden Retriever");
    cout<<d.getDesc();
    return 0;
}
```

- The class Dog has all the data members and methods of the Animal, as well as the breed name member and a getBreedName method.

Another example

```
#include <iostream>
using namespace std;

class Shape
{
protected:
    float width, height;
public:
    void set_data (float a, float b)
    {
        width = a;
        height = b;
    }
};

class Rectangle: public Shape
{
public:
    float area ()
    {
        return (width * height);
    }
};
```

```
class Triangle: public Shape
{
public:
    float area ()
    {
        return (width * height / 2);
    }
};

int main (){

    Rectangle rect;
    Triangle tri;
    rect.set_data (5,3);
    tri.set_data (2,5);
    cout << rect.area() << endl;
    cout << tri.area() << endl;
    return 0;
}
```

Polymorphism

- One object can have many types.
- If a function expects an Animal object, a Dog object can be given because any Dog is also an Animal.

Multiple Inheritance

- A class can have multiple based classes.

```
class Cat : public Animal, public HealthInsuredPet{
    ... };
```

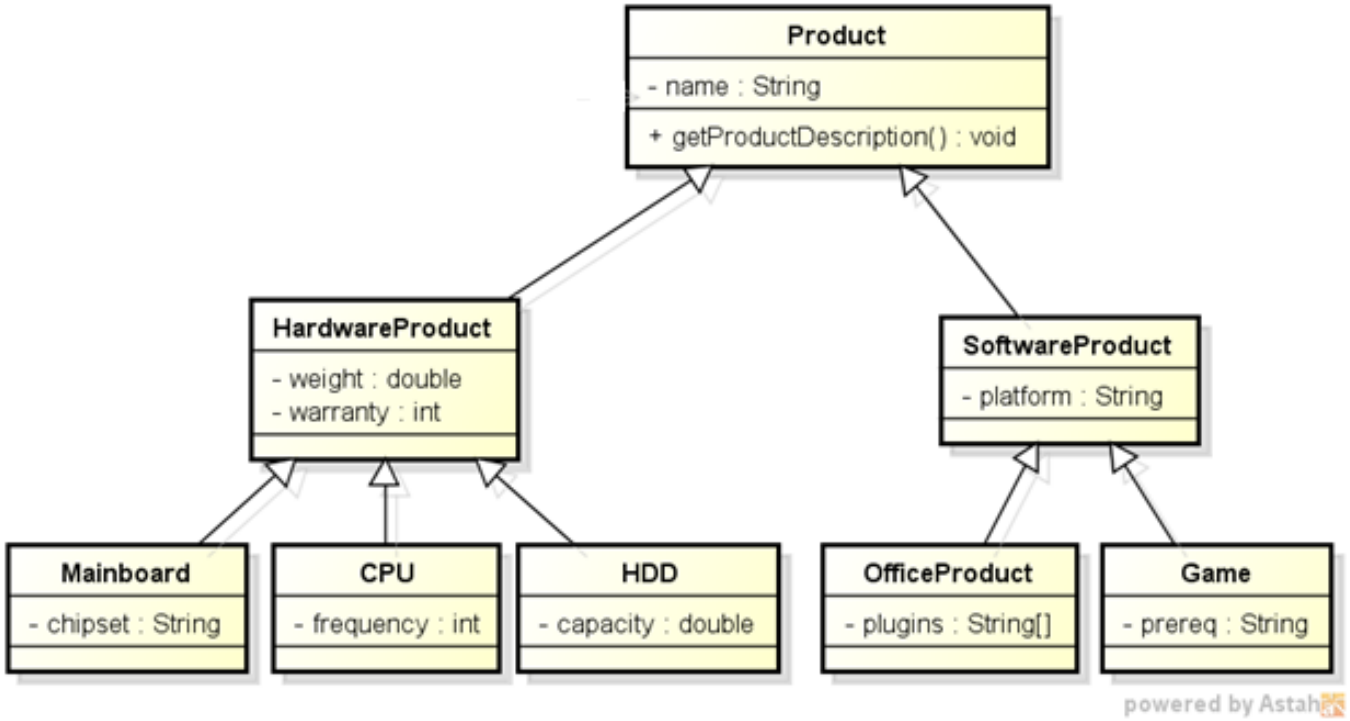
Multiple Inheritance

- A class can have multiple based classes.

```
class Cat : public Animal, public HealthInsuredPet{
    ... };
```

Exercises

- 1. Implement the following situation:



Vectors

Can be used to dynamically allocate an array.

```
#include<iostream>
#include<string>
#include<vector>
using namespace std;

int main()
{
    vector<string> colour;

    colour.push_back("Blue");
    colour.push_back("Red");
    colour.push_back("Orange");
    colour.push_back("Yellow");
    //The operator sizeof has the return type of std::size_t
    for (std::size_t i=0; i<colour.size(); i++)
        cout << colour[i] << "\n";
}
```

Vectors used in the previous problem with Animal and Dog

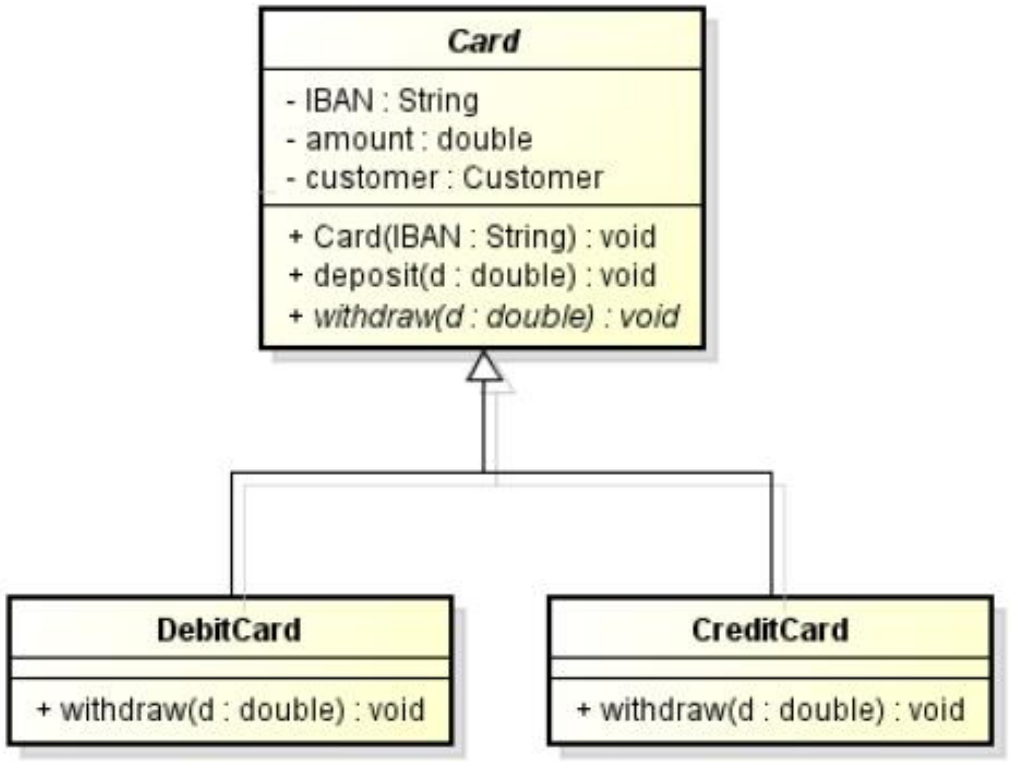
```
int main()
{
    Dog d("Rolly", 5, "Golden Retriever");
    cout<<d.getDesc();
    vector<Animal> animals;

    animals.push_back(d);

    //The operator sizeof has the return type of std::size_t
    for (std::size_t i=0; i<animals.size(); i++)
        cout <<endl<< animals[i].getDesc() << "\n";
    return 0;
}
```

Exercises

- Implement the following situation:



Homework

- Implement the following situation: