Nile University

Data Structures and Algorithms

Episode Two

Author: Ahmed Khaled

Nile University

Remember Last Episode

Material & Code Files

Github Repo

GOAL

```cpp
int main() {
    Person<int>*    p1 = new Person<int>(990745, "Mohammed", 22);
    Person<string>* p2 = new Employee("accd550", "Adam", 26, 2000);
    Person<int>*    p3 = new Student(23100077, "Rawan", 19, 4.0);

    printPersonDetails(*p1);
    printPersonDetails(*p2);
    printPersonDetails(*p3);

    delete p1;
    delete p2;
    delete p3;

    return 0;
}
```

Output:

```
Role: Person
Name: Mohammed, Age: 22, National ID: 990745
Role: Person
Name: Adam, Age: 26, National ID: accd550
Role: Person
Name: Rawan, Age: 19, National ID: 23100077
```

Agenda

- **QUIZ**
- A Class with Getters & Setters
- Inherit a Base Class, **reusing** parent constructor.
- Access Modifiers, public, private, **protected**.
- Overloading and Overriding!
- Class Template (Generic Class) and Function Template
- Header Files .h and Implementation Files .c++

Quiz

Write a C++ program that defines a Car class with a constructor to initialize its brand and year, a destructor, and a member function to display the car's details.

In the main() function, create a Car object, display its information, and observe the constructor and destructor messages.

Nile University

Person Class, with Getters and Setters

```cpp
class Person
{
private:
    string name;
    string id;
    int age;
public:
    // constructors
    Person(): name("NONE"), age(0), id("NONE")  {}
    Person(string id, string name, int age): name(name), age(age), id(id) {}
    // setters and getters
    void setName(string name){this->name=name;}
    void setid(string id){this->id=id;}
    void setage(int age){this->age=age;}
    string getid(){return this->id;}
    string getname(){return this->name;}
    int getage(){return this->age;}
    // member functions
    void display_info(){cout<<"Name: "<<this->name<<" age:
"<<this->age<<endl;}
    string role(){return "Person";}
    // constructor
    ~Person(){cout<<"Object "<<this->id<<" has been removed from
memory"<<endl;}
};
```

```cpp
int main(){
    Person someperson("29907141401518" , "Ahmed",
24);
    someperson.setName("Mohammed");
    someperson.display_info();
    return 0;
}
```

Output:

```
Name: Mohammed age: 24
Role: Person
Object 29907141401518 has been removed
from memory
```

Inherit a Base Class, reusing parent constructor

```
class Employee: public Person{
   public:
        Employee(): Person() {};
};
```

```
int main(){
   Employee some_employee;
   some_employee.display_info();
   return 0;
}
```

Output:

```
Name: NONE age: 0
Object NONE has been removed from memory
```

Nile University

Overriding, customize base class function, same signature

```
class Person
{
      protected:
         string name;
         string id;
         int age;
      ...
}

class Employee: public Person{
   private:
       double salary;
   public:
       // overriding constructor(s)
       Employee(): Person(), salary(0.0) {};
       Employee(string id, string name, int age, double salary): Person(id, name, age),
salary(0.0) {};
       // Setters and getters
       void setsalary(double salary){this->salary=salary;}
       double getsalary(){return this->salary;}
       // overriding member function
       string role(){return "Employee";}
       void display_info(){cout<<"Name: "<<this->name<<" age: "<<this->age<< " salary
"<<this->salary<< endl;}
};
```

```
int main(){
   Employee someemployee ("A102", "Abdo", 30,
10000);
   cout<< someemployee .role() << endl;
   someemployee .display_info ();
   return 0;
}
```

Output:

```
Employee
Name: Abdo age: 30 salary 0
Object A102 has been removed
from memory
```

Access Modifiers private, protected, public

Access Modifiers

Access specifier meaning:

- Base class's private members can not be accessed in a derived class.
- Base class's protected members can be accessed in a derived class.
- Base class's public members can be accessed from anywhere.

Will only focus on **public** inheritance.

Nile University

Template Class, make class type independent

```
template <class T>
class Box{
   private:
       T content;
   public:
       // constructor
       Box(){this->content=-1;};
       Box(T content){this->content=content;};
       // getters and setters
       void setcontent(T content){this->content=content;}
       T getcontent(){return this->content;}
       // member function(s)
       void show_content(){cout<< this->content << endl;}
       // destructor
       ~Box(){};
};
```

```
int main(){
   Box<string> box1("Help");
   Box<int> box 2(1000);
   Box<double> box 3(9.3);
}

   box1.show_content();
   box2.show_content();
   box3.show_content();

   return 0;
}
```

Output:

```
Help
1000
9.3
```

Template Function, make function type independent

```
template<typename T>
bool compare_boxes(Box<T> b1, Box<T> b2){
   return b1.getcontent() == b2.getcontent();
}

int main(){
   Box<string> box1("Help");
   Box<int> box 2(10);
   Box<double> box 3(9.3);
   Box<int> box 4(11);

   if (! compare_boxes(box2, box4)){
       cout<<"Boxes have different CONTENT"<<endl;
   }
   return 0;
}
```

Output:

```
Boxes have different CONTENT
```

Nile University

Thank You