Инкапсулација, наслеђивање и полиморфизам

1. Инкапсулација

Инкапсулација је један од основних концепата ООП. Инкапсулација обједињује податке и функције чланице класе чинећи их сигурним и недоступним изван објекта уколико за то постоји потреба. Она на известан начин изолује податке и функције чланице класе од спољашњег дела програма. У Це++ језику особине инкапсулације доступне су кроз употребу кориснички дефинисаних типова података који се називају класама. Класе разликују три типа права приступа: приватни (енгл. *private*), заштићени (енгл. *protected*) и јавни (енгл. *public*). Ако се експлицитно не дефинише тип права приступа подразумеван је приватни тип приступа. Пример:

```
class Student
{
      string m_name;                // private by default
      int m_id;                     // private by default
public:
      Student(void);
      ~Student(void);
      string getName();
      void setName(string name);
      int getId();
      void setId(int id);
};
```

m_name и ***m_id*** су приватни чланови класе Student. Ово значи да њима могу приступити само други чланови класе, фуннкције чланице, док су за спољашњи део програма они недоступни (невидљиви).

Да би омогућили да чланови класе буду доступни изван класе потребно је декларисати их као јавне чланове односно иза резервисане речи *public*. Сва поља класе дефинисана иза резервисане речи *public* видљива су изван класе и њима могу приступити и друге слободне функције.

Заштићени тип права приступа, *protected*, дефинише иста права приступа као и приватни тип изузев у случају изведених класа. Поља класе која су дефинисана као заштићена видљива су унутар класе као и у изведеним класама које приликом извођења из основне класе користе квалификаторе *public* и *protected*.

2. Наслеђивање

Наслеђивање такође представља један од основних концепата ОО парадигме. Наслеђивање нам омогућује лакше програмирање и писање нових програма, лакше

одржавање и модификовање као и вишекратно коришћење већ постојећих програмских модула. Начелно наслеђивање омогућује преузимање садржаја неких класа у дату класу уз могућност измене постојећих чланова и додавање нових. Терминолошки класа (или класе) од које се преузима садржај зове се предак, наткласа или основна класа. Класа која преузима садржај назива се потомак, потклаcа или изведена класа. У основи, наслеђивања је однос и веза између објеката. На пример, Gmail корисник је особа, Google+ корисник је Gmail корисник, па према томе Google+ корисник је уједно и особа.

2.1 Основне и изведене класе

Класа може бити изведена из више основних класа и у том случају она наслеђује све њихове податке и функције чланице. Приликом дефинисања изведене класе наводи се листа основних класа које нова класа жели да наследи. Листа основних класа садржи имена једне или више класа и дефинише се на следећи начин:

```
class derived-class: access-specifier base-class
```

У дефиницији изведене класе ***access-specifier*** представља *public*, *private* или *protected* квалификатор који одређује права приступа атрибутима основне класе. Уколико квалификатор права приступа није дефинисан он узима подразумевану вредност *private*. Анализирати приложени пример (доле у наставку).

3. Полиморфизам

По свом значењу реч полиморфизам односи се на нешто што има више форми или облика. Типично, у контексту ООП парадигме, полиморфизам се испољава у хијерархији класа које су међусобно повезане наслеђивањем. Поједностављено, полиморфизам у Це++ језику омогућава да различита имлементација одређене функције чланице буде позвана у зависности од типа објекта који позива ту конкретну функцију чланицу. Посматрајмо следећи пример:

```
#include <iostream>

using namespace std;

class Shape
{
public:
      ~Shape();
      void draw();
```

```
};

class Circle : public Shape
{
public:
	~Circle();
	void draw();
};

Shape::~Shape()
{
	cout << "Shape destructor" << endl;
}

void Shape::draw()
{
	cout << "Shape::draw" << endl;
}

Circle::~Circle()
{
	cout << "Circle destructor" << endl;
}

void Circle::draw()
{
	cout << "Circle::draw" << endl;
}

int main()
{
	Shape *shape = new Circle;
	Shape->draw();
	return 0;
}
```

Превести и покренути програм. Излаз је следећи:

```
Shape::draw
Shape destructor
```

Разлог зашто се програм извршио на овакав начин налази се у томе како је преводилац разрешио коју имплементацију деструктора и функције draw() да позове. У овом случају позиви функција разрешени су у току превођења програма и то на основу типа објекта, имена и аргумената функције. Овакав начин одређивања позива функција зове се статичко разрешивање (енгл. *static resolution*) или статичко повезивање (енгл. *static linkage*) – позиви функција одређени су пре почетка програма. Такође овај начин разрешивања зове се и рано повезивање (енгл. *early binding*). Изменимо сада приложени код тако што ћемо само испред декларације функције draw() у класи Shape додати резервисану реч Це++ језика **virtual**.

Превести и покренути програм. Излаз је следећи:

```
Circle::draw
Shape destructor
```

Као што се можемо уверити овог пута је уместо функције чланице класе Shape позвана функција чланица класе Circle. У овом случају позив функције одређује се у току извршавања програма тако што се посматра адреса објекта који позива функцију а не његов тип. У показивачу **shape* налази се адреса објекта класе Circle па приликом позива функције draw() биће позвана функција чланица класе Circle а не класе Shape. Исти проблем односи се и на деструктор класе. Поновите претходну процедуру и испред декларације деструктора у основној класи Shape додајте реч *virtual.* Шта је резултат извршења програма?

Ово је уопштени начин како се полиморфизам примењује у Це++ језику. Могуће је дефинисати више изведених класа које садрже исте функције као основна класа, а при томе свака од изведених класа има јединствену имплемнтацију истих.

3.1 Виртуелна функција

Виртуелна функција декларише се у основној класи употребом резервисане речи **virtual**. Приликом дефинисања тела виртуелне функције унутар изведене класе преводиоцу се шаље сигнал да се не ради о подразумеваном статичком разрешивању позива функција већ о нечем другом. У овом случају преводилац ће користити динамичко (енгл. *dynamic linkage*) или касно повезивање (енгл. *late binding*) – која функција ће бити позвана биће одлучено у току извршавања програма а не на основу типа објекта.

3.2 Чиста виртуелна функција

Некада постоји потреба да се у основној класи изостави било каква дефиниција виртуелне функције чланице да ли из разлога што не постоји логичка оправданост да се дефинише било каква функционалност или пак зато што се у потпуности даје могућност изведеним класама да прилагоде дефиницију својим потребама. У Це++ језику могуће је декларисати чисту виртуелну функцију на следећи начин:

```
virtual method-name() = 0;
```

Чиста виртуелна функција не поседује тело функције, оно је у потпуности изостављено. Класа која поседује чисту виртуелну функцију постаје апстрактна класа (или интерфејс). Апстрактне класе могу се посматрати као недовршене или некомплетне класе из разлога што чисте виртуелне функције немају своју имплементацију – то се очекује да уради изведена класа. Из претходно поменутог произилази да се апстракне класе не могу инстанцирати, односно није могуће направити објекат апстрактне класе. Задатак: Преправити приложени пример тако да класа Shape буде апстрактна класа.

Вежба 1

1. Направити апстрактну класу **MyShape** која садржи два атрибута **x** и **y,** и једну чисту виртуелну функцију **getArea()**.

2. Направити две изведене класе **MyCircle** и **MyRectangle** које наслеђују класу **MyShape,** и изведену класу **MySquare** која наслеђује класу **MyRectangle**. Све класе треба да имају одговарајуће конструкторе и функције чланице. Атрибути основних класа треба да буду видљиви само изведеним класама.

3. Проверити решење користећи *main* функцију која је приложена у пројекту.

Вежба 2

Написати програм који ће рачунати обим и површину, на основу унетих параметара. Кориснику омогућити да унесе димензије:

- *Circle radius* – једно поље за полупречник
- *Rectangle width and height* – по једно поље за ширину и висину
- *Square width* – једно поље за ширину,

Као и тип фигуре за који се врши рачунање:

- *Circle*
- *Rectangle*
- *Square*

Резултат исписати на конзоли.

Додатни задатак:

Проширити задатак 2 тако да ради са улазном и излазном датотеком.