**Ringkasan Kisi-Kisi Essay UAS Object Oriented Programming**

Materi : Inheritance, Composition, Polymorphism.

1. Inheritance adalah suatu proses atau mekanisme untuk membuat class baru dari class yang sudah ada sebelumnya atau mendefinisikan class baru agar lebih spesifik dengan mewariskan sifat class yang sudah ada.
2. Keuntungan inheritance :

- Menghindari penggandaan data yang sama 🡪 tidak perlu membuat data lagi di class yang baru jika sudah ada di class yang pernah dibuat. Cukup diturunkan saja.

1. Jenis inheritance berdasarkan banyaknya class :

- Single inheritance 🡪 pewarisan yang jumlah base classnya tunggal/satu, tetapi class turunan bisa lebih dari satu.



- Multiple inheritance 🡪 class yang diturunkan lebih dari satu class yang berbeda (jumlah base class lebih dari satu). C++ mendukung jenis inheritance ini.



- Virtual Multiple Inheritance 🡪 class yang diturunkan lebih dari satu class yang berbeda. Bedanya dengan multiple inheritance, beberapa diantara base class tersebut merupakan turunan dari base class yang sama.

1. Jenis inheritance berdasarkan access specifiernya :

- Public inheritance

- Protected inheritance

- Private inheritance

1. Jenis inheritance berdasarkan hubungannya :

- Direct inheritance 🡪 sebagai contoh ada 3 class A, B, C. Class C merupakan turunan dari class B, sementara B turunan dari class A. Yang merupakan direct inheritance adalah Class C - Class B, dan Class B – Class A.



- Indirect inheritance 🡪 sebagai contoh ada 3 class A, B, C. Class C merupakan turunan dari class B, sementara B turunan dari class A. Yang merupakan indirect inheritance adalah class C – Class A.

1. Kita bisa menginisialisasi base class data member dari class turunannya dengan cara membuat constructor di base class menggunakan parameter. Pada saat membuat class turunan, constructor base class akan dipanggil. Contoh code :

#include<iostream>

using namespace std;

class rectangle{

protected:

 int length,width;

public:

 void setLength(int length){

 this->length=length;

 }

 void setWidth(int width){

 this->width=width;

 }

 int getLength(){

 return length;

 }

 int getWidth(){

 return width;

 }

 void area();

 rectangle(int length,int width) : length(length),width(width){

 cout<<"Initialize rectangle length : "<<length<<endl;

 cout<<"Initialize rectangle width : "<<width<<endl;

 }

};

void rectangle::area(){

 cout<<"Area = "<<length\*width;

}

class block : public rectangle{

private:

 int height;

public:

 void setHeight(int height){

 this->height=height;

 }

 int getHeight(){

 return height;

 }

 void area();

 void volume();

 block() : rectangle(1,1){

 height=0;

 width=0;

 length=0;

 cout<<"Initialize block height : "<<height<<endl;

 cout<<"Initialize block width : "<<width<<endl;

 cout<<"Initialize block length : "<<length<<endl;

 }

};

void block::area(){

 cout<<"Area = "<<(length\*width)+(width\*height)+(height\*length);

}

void block::volume(){

 cout<<"Volume = "<<height\*length\*width;

}

int main(){

 block a;

 a.setLength(7);

 a.setWidth(6);

 a.setHeight(10);

 a.area();

 cout<<endl;

 a.volume();

 cin.get();

 return 0;

}

Output yang dihasilkan :

Initialize rectangle length : 1

Initialize rectangle width : 1

Initialize block height : 0

Initialize block width : 0

Initialize block length : 0

Area = 172

Volume = 420

1. Yang tidak ikut di inherit apabila ada inheritance :

- Constructor dan Destructor

- Operator=() member

- Friend function

Dikarenakan functions-functions ini adalah class-specific. Artinya functions tersebut adalah milik masing-masing class tertentu. Misalnya constructor, di class turunan, ia memiliki constructornya sendiri.

1. Operator yang di overload di super class tapi tidak di overload di class turunan disebut non-member operator di class turunan.
2. Embedded object sering juga disebut nested object. Ini merupakan keadaan dimana pada suatu class dibuat object dari class lain. Perbedaan nested object dengan inheritance adalah :

- Nested object : akses di main harus ada akses class A di B.

- Inheritance : dapat diakses langsung dari object dari class turunannya.

- Contoh nested object :

#include<iostream>

using namespace std;

class kendaraan{

private:

 long long int harga;

public:

 void setHarga(long long int harga){

 this->harga=harga;

 }

 long long int getHarga(){

 return harga;

 }

};

class mobil{

public:

 kendaraan x;

 long long int hitung\_pajak(long long int harga){

 return 0.02\*harga;

 }

};

int main(){

 mobil a;

 a.x.setHarga(355000000);

 cout<<a.hitung\_pajak(a.x.getHarga())<<endl;

 cin.get();

 return 0;

}

Output :

7100000

1. Masalah pada multiple inheritance 🡪 diamond problem.



- Repeated inheritance adalah multiple inheritance dengan common grandparent. Repeated inheritance dibagi lagi menjadi 2 yaitu replicated inheritance dan shared inheritance.

- Replicated inheritance adalah sebuah feature yang diwarisi dari common ancestor dengan nama yang sama menjadi lebih dari 1 buah feature dalam current class. Dalam hal ini, feature yang diwarisi muncul lebih dari 1 kali.

- Shared inheritance adalah sebuah feature yang diwarisi dari common ancestor dengan nama yang sama menjadi hanya 1 buah feature dalam current class. Dalam hal ini feature yang diwarisi hanya muncul 1 kali.

1. Masalah lain pada multiple inheritance :

- Ambigu secara semantik karena nama yang sama di parent class.

- Nama yang sama bisa digunakan untuk operasi yang berbeda di base class.

- Solusi : redefine function yang ambigu di class baru dan gunakan function name yang lengkap.

1. Composition adalah cara untuk mengkombinasikan simple object menjadi sebuah complex object. Sering juga disebut dengan “has a” relationship. Pada composition apabila salah satu part pada complex object mati, maka seluruh complex object tersebut juga akan mati. Sebagai contoh class mobil memiliki engine a, wheel b, dan suspension c.

#include<string>

#include<iostream>

using namespace std;

class engine{

protected:

 int capacity;

 string type;

public:

 void setCapacity(int capacity){

 this->capacity=capacity;

 }

 int getCapacity(){

 return capacity;

 }

 void setType(string type){

 this->type=type;

 }

 string getType(){

 return type;

 }

 engine(){

 cout<<"Engine created"<<endl;

 }

 ~engine(){

 cout<<"Engine deleted"<<endl;

 }

};

class wheel{

protected:

 int size;

public:

 void setSize(int size){

 this->size=size;

 }

 int getSize(){

 return size;

 }

 wheel(){

 cout<<"Wheel created"<<endl;

 }

 ~wheel(){

 cout<<"Wheel deleted"<<endl;

 }

};

class suspension{

protected:

 string jenis;

public:

 void setJenis(string jenis){

 this->jenis=jenis;

 }

 string getJenis(){

 return jenis;

 }

 suspension(){

 cout<<"Suspension created"<<endl;

 }

 ~suspension(){

 cout<<"Suspension deleted"<<endl;

 }

};

class mobil{

public:

 engine a;

 wheel b;

 suspension c;

 mobil(){

 cout<<"Mobil created"<<endl;

 }

 ~mobil(){

 cout<<"Mobil deleted"<<endl;

 }

};

int main(){

 {

 mobil x;

 }

 cin.get();

 return 0;

}

Output :



1. Aggregation adalah hubungan antar 2 class yang juga sering disebut dengan “has-a” relationship. Perbedaannya dengan composition adalah pada aggregation adalah pada complex object jika kehilangan salah 1 partnya, maka object tersebut tetap bisa hidup. Sebagai contoh : department dengan guru.

#include<string>

#include<iostream>

using namespace std;

class Teacher{

private:

 string nama;

public:

 Teacher(string nama){

 cout<<"Teacher's built"<<endl;

 }

 ~Teacher(){

 cout<<"Teacher's deleted"<<endl;

 }

 string GetName() {

 return nama;

 }

};

class Department{

private:

 Teacher \*ptrTeacher;

public:

 Department(Teacher \*ptrTeacher=NULL){

 cout<<"Department's built"<<endl;

 }

 ~Department(){

 cout<<"Department's deleted"<<endl;

 }

};

int main(){

 Teacher \*a = new Teacher("Bob");

 {

 Department dept(a);

 }

 cin.get();

 return 0;

}

Output :



1. Class template :

- Dikenal juga sebagai generic class atau generic generator atau parameterized type.

- Biasanya digunakan untuk define class model.

- Syntax : template<class T> class X{...}

- Kita bisa mempunyai tipe data dengan operasi dasar misal insert, delete.

- Contoh class template :

#include<iostream>

using namespace std;

template<class T>

class segitiga{

protected:

 T alas;

 T tinggi;

public:

 void setAlas(T alas){

 this->alas=alas;

 }

 T getAlas(){

 return alas;

 }

 void setTinggi(T tinggi){

 this->tinggi=tinggi;

 }

 T getTinggi(){

 return tinggi;

 }

 T luas(T alas,T tinggi){

 return 0.5\*alas\*tinggi;

 }

};

template<class T>

class persegi:public segitiga<T>{

public:

 T luas(T alas,T tinggi){

 return alas\*tinggi;

 }

};

int main(){

 segitiga<int> satu;

 satu.setAlas(65);

 satu.setTinggi(30);

 cout<<"Luas segitiga satu = "<<satu.luas(satu.getAlas(),satu.getTinggi())<<endl;

 segitiga<double> dua;

 dua.setAlas(5.6);

 dua.setTinggi(4.4);

 cout<<"Luas segitiga dua = "<<dua.luas(dua.getAlas(),dua.getTinggi())<<endl;

 persegi<double> tiga;

 tiga.setAlas(5.6);

 tiga.setTinggi(4.4);

 cout<<"Luas persegi tiga = "<<tiga.persegi::luas(tiga.getAlas(),tiga.getTinggi())<<endl;

 persegi<int> empat;

 empat.setAlas(65);

 empat.setTinggi(30);

 cout<<"Luas persegi empat = "<<empat.persegi::luas(empat.getAlas(),empat.getTinggi())<<endl;

 cin.get();

 return 0;

}

Output :



1. STL (Standard Template Library) :

- Komponen : containers, iterators, dan algorithms.

- Cara penggunaan : memanggil dengan #include. Contoh #include<algorithm>.

- STL diimplentasikan sebagai class template yang memberikan flexibilitas dalam tipe yang didukung sebagai elemen.

- STL memberikan template definitions untuk programming tasks.

- STL mengizinkan program untuk ditulis sehingga code tidak tergantung pada underlying container, yang juga disebut sebagai Generic Programming.

1. Generic programming adalah program yang ditulis sekali dan dapat dipakai berulang-ulang untuk data beragam tanpa harus mengubah inti dari algoritma.
2. OOP vs Generic Programming :

- OOP :

* Actual parameter harus diturunkan dari formal parameter.
* Harus menggunakan virtual member, dengan associated overhead.
* Dapat menunda pemilihan function sampai run-time.

- Generic Programming :

* Parameter dibutuhkan hanya untuk mengimplementasikan operasi.
* Tidak ada run-time overhead, memungkinkan inlining.
* Tipe harus diketahui pada saat compile-time.

- Kedua paradigma adalah orthogonal.

- Orthogonality dalam bahasa program adalah kemampuan untuk menggunakan berbagai fitur bahasa program dalam berbagai kombinasi dengan hasil yang konsisten.

1. Polymorphism :

- Poly 🡪 multiple / banyak

- Morph 🡪 bentuk / forms

- Polymorphism 🡪 sebuah object yang dapat melakukan banyak fungsi. Sebagai contoh jika object menunjuk ke class dasar maka ia akan berperilaku seperti class dasar, jika object menunjuk ke class turunan, ia akan berperilaku seperti class turunan.

1. Ada 3 jenis polymorphism :

- Ad-Hoc polymorphism 🡪 dicapai dengan mekanisme overloading. Merupakan compile-time polymorphism.

- Parameterized polymorphism 🡪 untuk mengeksekusi kode yang sama untuk semua tipe. Dicapai dengan menggunakan template. Syntax umum template : template<class T>. Juga merupakan compile-time polymorphism.

- Subtype polymorphism 🡪 pemanfaatan pointer dari base class untuk mengakses derive class. Polymorphism ini membutuhkan adanya inheritance. Sifat-sifat polymorphism ini :

* Dynamic binding, menggunakan keyword virtual.
* Run-time polymorphism.
* Dicapai dengan inheritance dan overriding. Menggunakan pointer.
* Reuse terbatas terhadap prototipe function.
* Prototype sama, tetapi tugas berbeda.
* Scope berbeda, class berbeda.
1. Contoh parameterized polymorphism (menggunakan class template) :

#include<iostream>

using namespace std;

template<class T>

class segitiga{

protected:

 T alas;

 T tinggi;

public:

 void setAlas(T alas){

 this->alas=alas;

 }

 T getAlas(){

 return alas;

 }

 void setTinggi(T tinggi){

 this->tinggi=tinggi;

 }

 T getTinggi(){

 return tinggi;

 }

 T luas(T alas,T tinggi){

 return 0.5\*alas\*tinggi;

 }

};

template<class T>

class persegi:public segitiga<T>{

public:

 T luas(T alas,T tinggi){

 return alas\*tinggi;

 }

};

int main(){

 segitiga<int> satu;

 satu.setAlas(65);

 satu.setTinggi(30);

 cout<<"Luas segitiga satu = "<<satu.luas(satu.getAlas(),satu.getTinggi())<<endl;

 segitiga<double> dua;

 dua.setAlas(5.6);

 dua.setTinggi(4.4);

 cout<<"Luas segitiga dua = "<<dua.luas(dua.getAlas(),dua.getTinggi())<<endl;

 persegi<double> tiga;

 tiga.setAlas(5.6);

 tiga.setTinggi(4.4);

 cout<<"Luas persegi tiga = "<<tiga.persegi::luas(tiga.getAlas(),tiga.getTinggi())<<endl;

 persegi<int> empat;

 empat.setAlas(65);

 empat.setTinggi(30);

 cout<<"Luas persegi empat = "<<empat.persegi::luas(empat.getAlas(),empat.getTinggi())<<endl;

 cin.get();

 return 0;

}

Output :



1. Contoh template sederhana :

#include<iostream>

#include<string>

using namespace std;

template<class T>

T jumlah(T a,T b){

 return a+b;

}

template<class T>

T kurang(T a,T b){

 return a-b;

}

template<class T, class U>

U jumlah\_campur(T a,U b){

 return a+b;

}

int main(){

 string nama1="Timothy Orvin ";

 string nama2="Clarisa Valencia";

 cout<<"Penjumlahan/Concatenate string = "<<jumlah(nama1,nama2)<<endl;

 int angka1=165;

 int angka2=333;

 cout<<"Penjumlahan int = "<<jumlah(angka1,angka2)<<endl;

 cout<<"Pengurangan int = "<<kurang(angka1,angka2)<<endl;

 double dbl1=65.3;

 double dbl2=100.34;

 cout<<"Penjumlahan double = "<<jumlah(dbl1,dbl2)<<endl;

 cout<<"Pengurangan double = "<<kurang(dbl1,dbl2)<<endl;

 cout<<"Penjumlahan campur angka1 dan dbl1 = "<<jumlah\_campur(angka1,dbl1)<<endl;

 cin.get();

 return 0;

}

Output :



1. Contoh subtype polymorphism :

#include<iostream>

using namespace std;

class kendaraan{

protected:

 long long int harga;

public:

 void setHarga(long long int harga){

 this->harga=harga;

 }

 long long int getHarga(){

 return harga;

 }

 virtual long long int hitung\_pajak(long long int harga){

 return 0;

 }

};

class mobil:public kendaraan{

public:

 long long int hitung\_pajak(long long int harga){

 return 0.02\*harga;

 }

};

class motor:public kendaraan{

public:

 long long int hitung\_pajak(long long int harga){

 return 0.01\*harga;

 }

};

int main(){

 kendaraan \*objptr;

 mobil a;

 motor b;

 cout<<"PAJAK MOBIL"<<endl;

 objptr=&a;

 objptr->setHarga(355000000);

 cout<<"Pajak mobil = "<<objptr->hitung\_pajak(objptr->getHarga())<<endl;

 cout<<endl;

 cout<<"PAJAK MOTOR"<<endl;

 objptr=&b;

 objptr->setHarga(12000000);

 cout<<"Pajak motor = "<<objptr->hitung\_pajak(objptr->getHarga())<<endl;

 cin.get();

 return 0;

}

Output :



1. Overriding terjadi ketika function di base class dengan nama sama, return type sama, dan parameter sama yang ditulis ulang di class turunannya. Mekanisme ini hanya terjadi di inheritance.
2. Virtual function adalah function yang dapat didefinisikan ulang oleh keturunannya. Keyword virtual digunakan untuk membuat binding menjadi dynamic. Secara default binding di C++ adalah static.
3. Jenis – jenis virtual function :

- Pure virtual function :

* Tidak ada body function.
* Pure virtual method digunakan untuk mendeklarasikan function prototype.
* Syntax : virtual void function()=0;
* Class dengan pure virtual function disebut juga abstract class.
* Pada abstract class, tidak bisa dibuat instance/object.

- Quasi virtual function :

* Mengandung minimal function. Biasanya adalah blank statement.
* Contoh :

class myClass{

 virtual void function(){;}

};

1. Static binding dan Dynamic binding :

- Static binding : early binding, compile-time, berdasarkan tipe termasuk pointer.

- Dynamic binding : late binding, run-time.

- Overloading dan overriding adalah static binding.

- Parametric polymorphism dapat berupa static atau dynamic binding.

- Pure polymorphism adalah dynamic binding.

1. Pure polymorphism :

- Menggunakan pointer dan reference.

- Menggunakan virtual function untuk mencapai dynamic binding.

- Terdapat abstract base class (pure virtual function).

1. Contoh pure polymorphism :

#include<iostream>

using namespace std;

class kendaraan{

protected:

 long long int harga;

public:

 void setHarga(long long int harga){

 this->harga=harga;

 }

 long long int getHarga(){

 return harga;

 }

 virtual long long int hitung\_pajak(long long int harga)=0;

};

class mobil:public kendaraan{

public:

 long long int hitung\_pajak(long long int harga){

 return 0.02\*harga;

 }

};

class motor:public kendaraan{

public:

 long long int hitung\_pajak(long long int harga){

 return 0.01\*harga;

 }

};

int main(){

 kendaraan \*objptr;

 mobil a;

 motor b;

 cout<<"PAJAK MOBIL"<<endl;

 objptr=&a;

 objptr->setHarga(355000000);

 cout<<"Pajak mobil = "<<objptr->hitung\_pajak(objptr->getHarga())<<endl;

 cout<<endl;

 cout<<"PAJAK MOTOR"<<endl;

 objptr=&b;

 objptr->setHarga(12000000);

 cout<<"Pajak motor = "<<objptr->hitung\_pajak(objptr->getHarga())<<endl;

 cin.get();

 return 0;

}

Output :

