**School of Computing**

**SECJ/SCSJ1023 Programming Technique 2**

**Tutorial**

**Name : \_\_\_\_\_\_NG JING ER\_\_\_\_\_\_\_\_ Matric No: \_\_A19EC0115\_\_\_**

**Answer all the Questions given.**

**NOTE: You are not allowed to use any resources while answering this Quiz.**

All answers must be answered using **your own words and example**.

1. Describe the difference between the following object relationships, and give examples for each relationship.
2. Difference between Composition versus Aggregation

**Aggregation represents an ownership relationship between the 2 classes and it is 1way relationship. And in aggregation the objects are independent which means if one of the parties being destroyed or deleted, the other party will not be affected and it will remain exists.**

**However, composition represents a strong ownership compare to aggregation. And for composition, the objects are strong dependent which means if one of the parties destroyed, the other party will be affected as well**.

1. Example Composition:

**Class Car- Class Engine**

1. Example Aggregation

**Class Person- Class Address**

1. Describe the difference between Overloading versus Overriding, and give examples for each concept.
2. Difference between Overloading versus Overriding.

**Overloading involves the methods in a class with same name but different parameter such as with different types or different numbers of parameter.**

**However, overriding involves dynamically binding by specifying the methods as virtual in parent class.**

1. Example Overloading. (Can show thru coding)

Overloading (methods below have the same name, but with different no. of parameter)

**class Setting{**

**private:**

**…;**

**public:**

**…;**

**void set (int r, int l) *// 2 arguments***

**{ room =r;**

**Level=l;**

**}**

**void set (int x) *//1 argument***

**{ cout<< “The room is not available.” << endl; }**

**};**

1. Example Overriding. (Can show thru coding)

**class House {**

**private:**

**…;**

**public:**

**virtual double getDeposit() *//define the method as virtual method***

**{ return 0; }**

**};**

**class Bungalow: public House {**

**private:**

**…;**

**public:**

**double getDeposit()**

**{ return 6\*rentRate;}**

**};**

1. Which concept (overloading or overriding) is used for polymorphism. Explain or give example.

**Overriding.**

**Polymorphism is that the methods from two class perform the same action differently. The method (in derived class with same name) will override the method in base class.**

**For example:**

**class Shape{**

**private:**

**…；**

**public:**

**virtual double getArea() *//The keyword virtual must be placed in parent class***

**{ return 0;}**

**};**

**class Rectangle: public Shape{**

**private:**

**double w;**

**double l;**

**public:**

**double getArea ()**

**{ return w\* l ;} *// Rectangle overrides the method getArea***

**};**

**class Triangle: public Shape {**

**private:**

**double width;**

**double length;**

**public:**

**double getArea()**

**{ return width\* length ;} *//Triangle overrides the method getArea***

**}；**

1. Given a set of classes as follows, draw UML diagram with attributes and methods to show the relationship between all classes.

Person, Apartment, Bungalow, Bedroom, Kitchen, LivingRoom, House, Address

|  |
| --- |
| **Kitchen** |
| -kwidth: double  -klength: double  -karea: double |
| +Kitchen()  +setWidth(\_width):void  +setLength(\_length):void  +getWidth():double  +getLength():double  +getKArea(): double |

|  |
| --- |
| **Person** |
| #name: string  -home:House |
| +Person()  +getName():string  +print():void  +lives(House\*):void |

|  |
| --- |
| **Address** |
| -street: string  -town: string |
| +Address()  +set(string,string):void  + getAddress():string |

|  |
| --- |
| **House** |
| **-**area:double  -address: Address\* |
| +House()  +getArea(): double  +getHouseAddress():string |

|  |
| --- |
| **Bedroom** |
| -brwidth: double  -brlength: double  -brarea: double |
| +Bedroom()  +setWidth(\_width):void  +setLength(\_length):void  +getWidth():double  +getLength():double  +getBArea(): double |

|  |
| --- |
| **Apartment** |
| #level:int |
| +Apartment()  +setLevel(int):void  +getLevel():int |

|  |
| --- |
| **LivingRoom** |
| **-**lrwidth: double  -lrlength: double  -lrarea: double |
| +LivingRoom()  +setWidth(\_width):void  +setLength(\_length):void  +getWidth():double  +getLength():double  +getLArea(): double |

|  |
| --- |
| **Bungalow** |
| #room:int |
| + Bungalow ()  +setRoom(int):void  +getRoom():int |

lives

1. Describe the following and give example thru coding.
2. Virtual method

**The keyword ‘virtual’ placed before the function in base class will redefine function in derived class.**

Example:

#include <iostream>

using namespace std;

class General{

public:

General(){}

virtual int getValue() const{return 0;} *//define the method using keyword virtual*

};

class Minor: public General {

private:

int value2;

public:

Minor(int b){ value2=b;}

int getValue() const {return value2;} *//the method become virtual in derived class*

};

void display(const General \*g){ cout << "Value is " << g->getValue() << endl; }

int main() {

General A;

Minor B(5);

display(&A); *// output will be 0(from base class)*

display(&B); *// output will be 5(from derived class)*

return 0;

}

1. Abstract class

**Abstract class is a class without any objects, therefore the compiler will generate error when we define an object of abstract class.**

**Abstract class contains one or more pure virtual functions.**

Example:

#include <iostream>

using namespace std;

class General{

public:

General(){}

virtual int getValue()=0; *//turns General to abstract class by changing method to pure virtual*

};

class Minor: public General {

private:

int value2;

public:

Minor(int b){ value2=b;}

int getValue() const {return value2;} *//the method become virtual in derived class*

};

int main() {

General a; *// if object from abstact class defined, error will be generated*

Minor b(5) ; *// if object from abstact class defined, error will be generated*

return 0;

}

1. Pure virtual function

**A pure virtual function is declaring methods in base class as virtual but without any definition.**

Example:

#include <iostream>

using namespace std;

class General{

public:

General(){}

virtual int getValue()=0; *// indicate the method =0*

};

class Minor: public General {

private:

int value2;

public:

Minor(int b){ value2=b;}

int getValue() const {return value2;}

};

int main() {

General a; *// error will be generated*

return 0;

}