****

**INSTRUCTIONS FOR CANDIDATES**

You are advised (but not required) to spend the first ten minutes of the examination reading the questions

**IMPORTANT:**

**The exam paper is provided in two versions according to the Course you are enrolled on.**

**Answer only the questions in the version corresponding to the programming language that you have been taught during the module:**

**1) Java Version (pages 2 - 4)**: BSc Computer Science, BSc Multimedia Computing, BSc Digital Media Development or BEng Software Engineering

**2) C++ Version (pages 5 - 6)**: BSc Computer Games Development

**Module Title: Object Oriented Programming**

**Module Code: 5COSC001W**

**Exam Period: January 2020**

**Time allowed: 2 hour**

**THIS PAPER MUST NOT BE TAKEN OUT OF THE EXAMINATION ROOM**

**DO NOT TURN OVER THIS PAGE UNTIL THE INVIGILATOR INSTRUCTS YOU TO DO SO**

**© UNIVERSITY OF WESTMINSTER, 2019-2020**

## JAVA Version

## Answer the 8 questions in this section only if you are enrolled on BSc Computer Science, BSc Multimedia Computing, BSc Digital Media Development or BEng Software Engineering

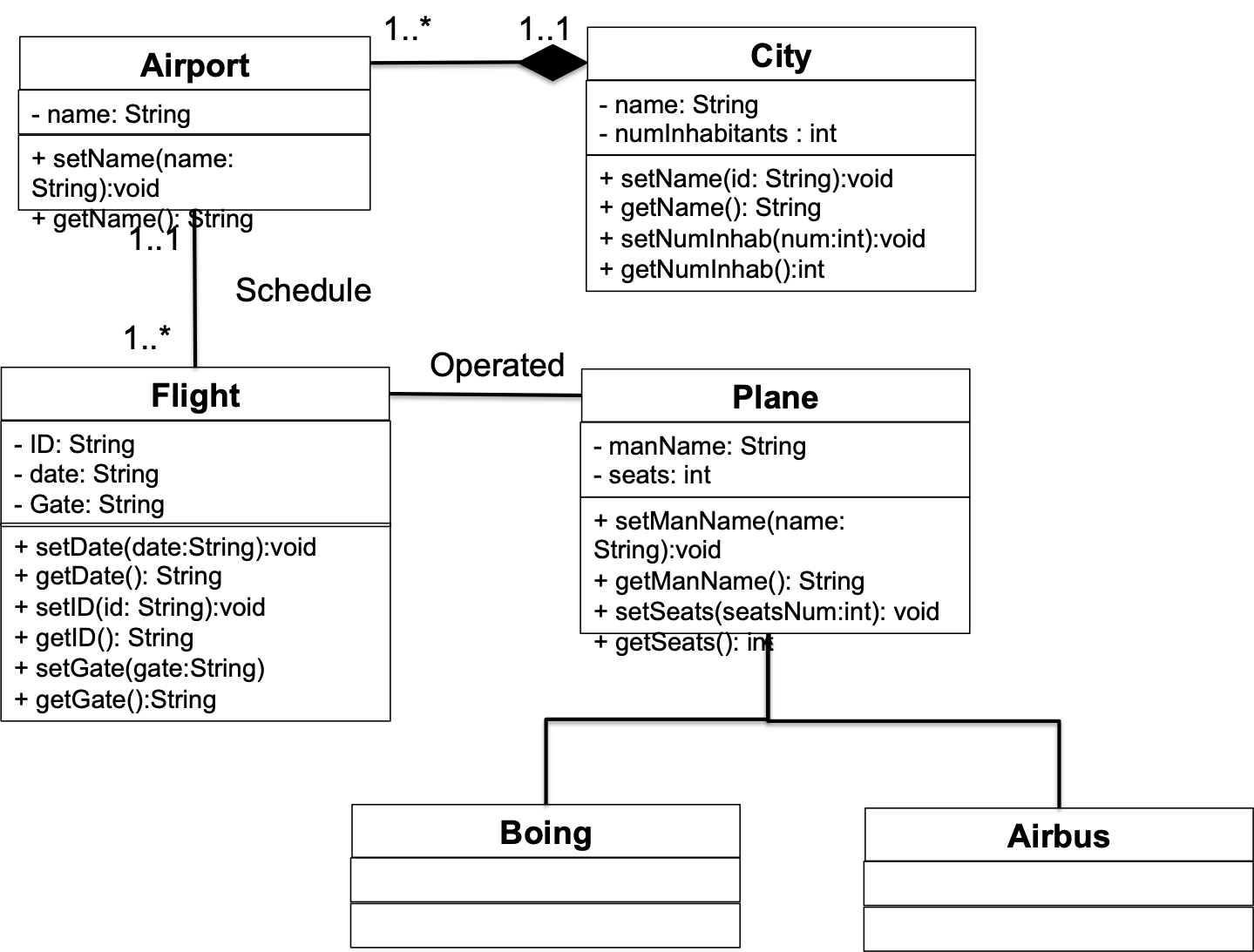
DO NOT REPLY to these questions if you are enrolled to BSc Computer Games Development

1) Draw a UML class diagram, of the problem presented below:

*Each airport has a name and it is built in a city. The city has a name and number of inhabitants.*

*Several flights are scheduled in the airport. Each flight is identified by a number, a date, a boarding time and a gate. The flight is operated with a plane, which can be either a boing or an airbus. Each plane has a number of seats and manufacturer’s name.*

*(12 points)*

**

*Sum up the following:*

*Max 6 points: identification of 6 blocks (1 point for each block)*

*Max 4 points: if all the relationships are correct and the composition has been identified*

*Max 2 points: if all multiplicities are correct and identified*

2) Briefly explain the meaning of the following terms. Use also practical examples to justify your answer:

* 1. Object

An object is

* Something tangible (person, pen, mug)
* Something that can be apprehended intellectually (Time, Date)
* An object has state, behavior, identity
* An object is an **Instance of a class.** (Ben is an instance of the class Person: a **specific object** that belongs to the class Person)
  1. Class

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties (instance variables) or behaviors (instance methods) that are common to all objects of one type. For example for the class Person we have:

* + State (attributes)
    - Name
    - Age
  + Behavior (operations)
    - Walks
    - Eats
  + Identity
    - His name
  1. Constructor

We need to call the constructor of the class in order to create instances of the class. For this we need the “new” keyword. Example: Account account1 = new Account(100).

When an object is created, its instance variables are initialised by a constructor.

The constructor MUST have the same name as the class name

A class can have more than one constructor

*(3 points each: tot 9 points)*

*(9 points)*

3) Suppose that class FootballPlayer and class BasketballPlayer are subclasses of class Player. Which of the following are legal?

1. FootballPlayer f = new Player();
2. Player p = new BasketballPlayer ();
3. BasketballPlayer b = new FootballPlayer ();

b

*(5 points)*

4) For all parts of this question, consider the following class definition that represents a pipe:

public class Pipe {

public double diameter; // from 0.5 to 5.0 cm

public double length; // from 10.0 to 100.0 cm

};

1. Provide a redesigned Pipe class that uses more appropriate access modifiers according the encapsulation principle.

public class Pipe {

private double diameter;

private double length;

};

*(5 points)*

1. Provide a constructor for the Pipe class that will initialise the instance variables to suitable (valid) start values.

public Pipe(){

this.diamter = 0.5;

this.length = 10;

}

Some guidance:

* Reduce one point -1 point if Pipe is declared as class
* Max 3 points If the student passed diameter and length as parameter and does not check if the values are valid

*(5 points)*

1. Provide signatures for set and get methods for one of the instance variables (you can choose which one).

public void setDiameter(int diameter);

public double getDiameter();

Some guidance:

* Max 3 points If the implementation has been provided

*(5 points)*

1. Write bodies for the methods for which you provided signatures in question (c) above. Note that for the set method, the implementation must prevent the two instance variables from being set to invalid values.

public void setDiameter(int diameter){

if(diameter >= 0.5 && diameter <=5.0)

this.diameter = diameter;

else

System.out.println (“invalid data”);

}

public int getDiameter(){

return diameter;

}

Some guidance:

* 4 points for get method, 4 point for set method
* In the set method, If the implementation does not prevent to set to invalid values gives max 2 points (over 4)
* If the set method is correct but there is no message printed on the screen reduce one point

*(8 points)*

1. Write a main method that instantiates the Pipe class and use the set and get methods you designed.

public class MainPipe{

public static void main(String[] args){

Pipe pipe = new Pipe();

pipe.setDiamter (2);

System.out.println(“The diameter is “ + pipe.getDiameter());

}}

*(5 points)*

5) What are abstract methods and classes in Java? Why they are used? Provide also an example.

Abstract methods are methods that are declared, with no implementation. It is possible to declare a method without defining it: public abstract void draw(int size);

Any class containing an abstract method is an abstract class. The class has to be declared with the keyword abstract: abstract class MyClass {...}

An abstract class is incomplete. It has “missing” method bodies. Thus, it is not possible to instantiate (create a new instance of) an abstract class. But the class can provide a constructor.

It is possible to inherit from an abstract class. If the subclass defines all the inherited abstract methods, it is “complete” and can be instantiated. If the subclass does not define all the inherited abstract methods, it must be abstract too. A class can be abstract even if it does not contain any abstract methods. This prevents the class from being instantiated.

Example:

public abstract class Shape{

private String color;

public Shape(){

color = “white”;

}

public abstract double getArea(); // abstract method

}

public class Square extends Shape{

private side;

public Shape(){

Super();

Side = 1;

}

public double getArea(){

return side\*side;

}

Some guidance:

* 10-9 points: Excellent answer where abstract method or abstract class are explained using examples.
* 8 points: very good answer but there are some inaccuracies
* 7-5 points: good reply but some concepts or code are not clear or not well explained
* 4-3 points: there are few good things, but the reply is vague
* 2-1 points: just an attempt

*(10 points)*

6) Concurrent data access can be regulated by synchronisation. Explain this problem and how the synchronisation can be solved in Java. You can explain using an example in Java code

Concurrent access to data can lead to data integrity problems. Specifically, if two sources attempt to update the same data at the same time, the result of the data can be undefined. The outcome is determined by how the scheduler schedules the two sources. Since the schedulers activities cannot be predicted, the outcome cannot be predicted. Java has a mechanism called synchronization to prevent this problem.

If one thread tries to read the data and another thread tries to update the same data leads to inconsistent state for the shared data. This can be prevented by synchronizing access to the data via Java synchronized keyword

public synchronized void update() {

...  
}

They can propose the deposit/withdraw example reported in the lecture.

Some guidance:

* 6-5 points: clear answer to the questions. The example is correct. (Still full points if the answer is excellent but no example)
* 4-3: Good reply but some concepts are confused
* 2-1: not clear reply but there is an attempt

*(6 points)*

7) Explain what is an Exception and the try-catch-throw mechanism in Java. You can use some examples in Java code.

Most of the OO languages provides features called exceptions. Exception is an indication of problem during execution.

Some typical causes of errors:

* + Memory errors (i.e. memory incorrectly allocated, memory leaks, “null pointer”) C++
  + File system errors (i.e. disk is full, disk has been removed)
  + Network errors (i.e. network is down, URL does not exist)
  + Calculation errors (i.e. divide by 0)
  + Array errors (i.e. accessing element –1)
  + Conversion errors (i.e. convert ‘q’ to a number)

In Java, C++, C#, Objective-C and Visual Basic, exceptions are handled by the keywords try, catch and throw.

Exception handling mechanism:

- Find the problem (try block : Hit the exception)

- Inform that an error has occurred (Throw the exception)

- Receive the error information (Catch the exception)

- Take corrective action (handle the exception)

Example:

try {

// possible nasty code

count = 0;

count = 5/count;

} catch (ArithmeticException e){

// code to handle the exception

System.out.println(e.getMessage());

count = 1;

}

System.out.println(“The Exception is handled”);

Some guidance:

* 6-5 points: clear answer to the questions. Also the example is provided and it is correct. (Still full points if the answer is excellent but no example)
* 4-3: Good reply but some concepts are confused
* 2-1: not clear reply but there is an attempt

*(6 points)*

8) Consider the following class named City:

public class City {

private String name;

private String country;

private int population;

public City(String name){

this.name = name;

}

public void setCountry(String country){

this.country = country;

}

public void setPopulation(int population){

this.population = population;

}

public String getCountry(){

return country;

}

public String getName(){

return name;

}

public int getPopulation(){

return population;

}

public String toString(){

return “ City: “ + name + “Population: “+ pointInLeague”;

}

}

1. Modify the implementation of the class City in order to provide a way to compare different Cities by their population. Write what you would change in the definition of the class and provide the implementation of the method to be used to compare. Note that you can use from java package java.lang.Comparable with the method public int compareTo(Tobj).

You don’t need to rewrite the whole class, just identifying and rewriting the sections that need changing would suffice.

public class City implements Comparable<City>{

private String name;

private String country;

private int population;

public City(String name){

this.name = name;

}

public void setCountry(String country){

this.country = country;

}

public void setPopulation(int population){

this.population = population;

}

public String getCountry(){

return country;

}

public String getName(){

return name;

}

public int getPopulation(){

return population;

}

public String toString(){

return “ City: “ + name + “Population: “+ pointInLeague”;

}

public int compareTo(City city){

return(this.population – city.population);

}

}

*(10 points)*

*implements Comparable<City>{ -> 5 points*

*method ->5 points*

1. Write a main method where three Cities are instantiated and stored in an array (or a list) of City. For each team, set the population.

public class Main{

public static void main(String[] args){

City c1 = new City(“Rome”);

c1.setPopulation(3000000);

city c2 = new City(“Berlin”);

c2.setPopulation (4000000);

City c3 = new city(“London”);

c3.setPopulation (9000000);

City[] list = new City[3];

list[0] = c1;

list[1] = c2;

list[2] = c3;

}

}

*(8 points)*

*3 Cities are created: 3 points*

*Array/list instantiation: 2 points*

*setPopulation for three Cities: 3 points*

*(8 points)*

1. Within the main method sort the array (or list) of City according to the criteria implemented in point a. and print the ordered list to the terminal.

To the previous class to add:

Arrays.sort(list);

System.out.println("Sorting of City list:\n"+Arrays.toString(list));

*Correct use of sort method: 3 points*

*Correct printing on the screen:3 points*

*(6 points)*

**\*END\***

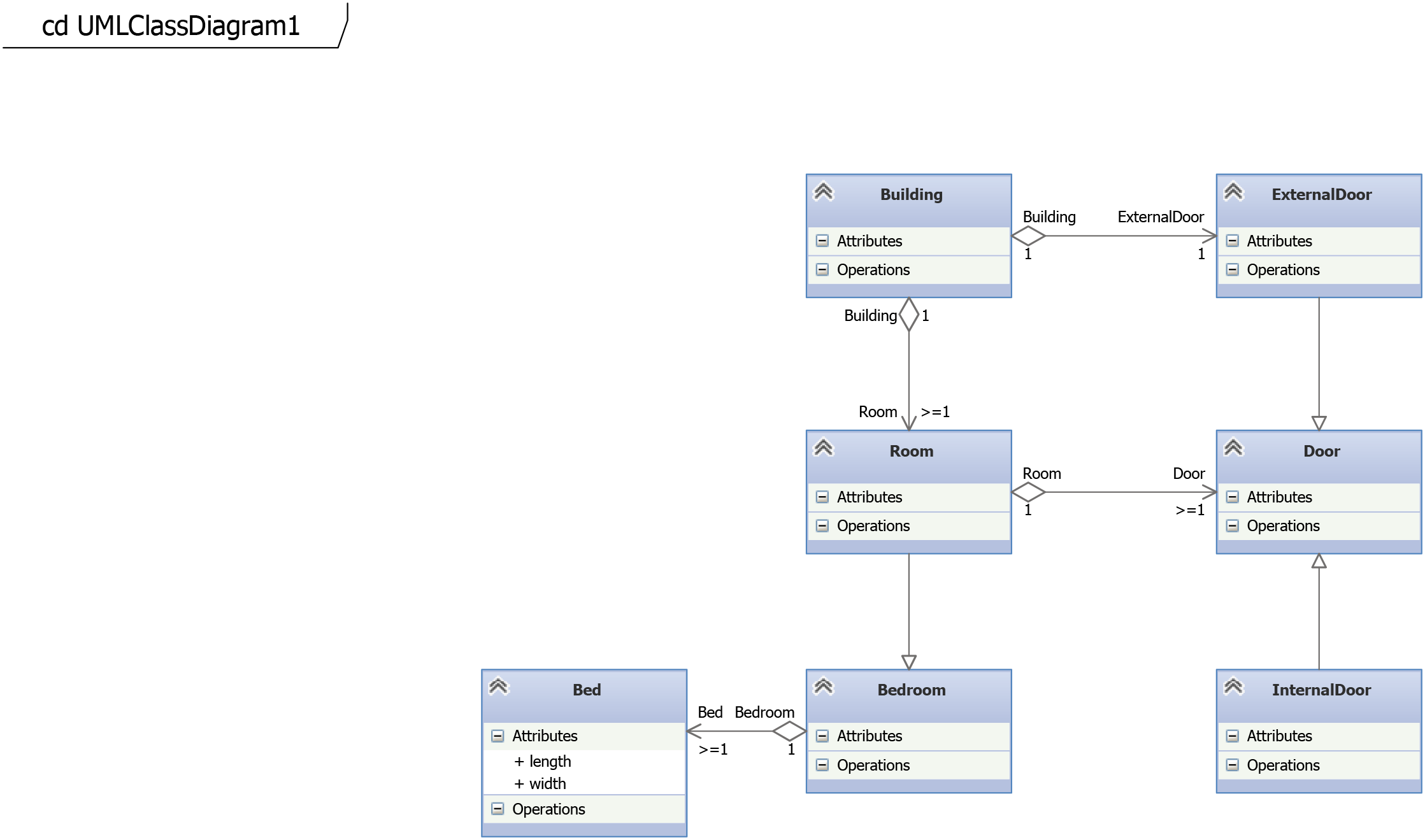
## C++ Version

## Answer the 8 questions in this section only if you are enrolled on BSc Computer Game Development

## DO NOT REPLY to these questions if you are enrolled to BSc Computer Science, BSc Multimedia Computing, BSc Digital Media Development or BEng Software Engineering

1) Draw a UML class diagram of the problem presented below. (12 points)

*A building has at least one room. Some rooms are bedrooms, and each bedroom has at least one bed. Each bed has a length and width, given in cm. Each room has at least one door. A door can be either internal or external. Each building has at least one external door.*



2) What is the Liskov Substitution Principle? (6 points)

The principle that objects of a subclass should be usable in place of objects of the parent class, i.e. inherited/overridden methods should satisfy the same assumptions/contract.

3) Create a Rectangle class. Each rectangle should have:

* a width and height (both integers) (4 points)
* a suitable constructor (5 points)
* a method getArea() returning the area of the instance (5 points)

The class should also:

* keep track of the total area of all rectangles created so far (6 points)
* have a method totalArea() returning this total. (6 points)

class Rectangle{

int width;

int height;

int total;

public:

Rectangle(int w, int h): width(w), height(h){

total += getArea();

}

int getArea(){

return width \* height;

}

static int totalArea(){

return total;

}

};

4) For this question, assume that we have included all necessary header files. Suppose we have two functions

void foo(int x, bool y) {…}

void bar(string &s) { … }

Complete the following function; the comments indicate what should happen.

void launch(){  
 string s = “sproing”;  
 // launch a thread running bar(s); (4 points)  
 thread t1(bar, s);  
 // launch another thread running foo(5,false); (4 points)  
 // it should run until the end of the program (4 points)  
 thread t2(foo, 5, false);

t2.detach();  
// Wait for the first thread to finish (4 points)  
t1.join();  
// Print “done” to standard output (2 points)  
cout << “done”;

}

5) The following questions revolve around implementing a Pet class.

a) Create an abstract Pet class with subclasses Spider and Snake. Each pet has a name (a string), for which there should be suitable getter and setter methods. They should also include constructors and a method numberOfLegs(), giving the number of legs based on the pet’s class. (12 points)

class Pet{

string name;

public:

string getName(){

return name;

}

void setName(string s){

name = s;

}

virtual int numberOfLegs()=0;

};

class Spider: public Pet{

Spider(string s): name(s){}

int numberOfLegs(){

return 8;

}

};

class Snake: public Pet{

Snake(string s): name(s){}

int numberOfLegs(){

return 0;

}

};

b) Write a function Pet \*getPet(string name, bool spider) which returns either a Spider (if spider is true) or a Snake (otherwise) with the given name. (6 points)

Pet \*getPet(string name, bool spider){

if(spider)

return new Spider(name);

else

return new Snake(name);

}

6) Describe the diamond problem. How can it be solved? (8 points)

C++ has multiple inheritance, i.e. a class C can have multiple parent classes. If two of these parents have a common superclass, any attributes in that superclass are contained twice in C. This leads to inconsistencies when these copies can have different values.

This can be solved using virtual inheritance.

7) Describe the purpose and main parts (classes, relations) of these design patterns:

a) Composite (6 points)

Composite is used for classes for which collections of instances should be able of the same kind of behaviour as individual instances. The main parts are:

* A component class (e.g. Shape)
* A number of subclasses (e.g. Rectangle, Circle, …)
* Among the subclasses, a Composite subclass which contains other components. Any method calls on a Composite object get forwarded to these components.

b) Strategy (6 points)

Strategy is used in case some tasks that a class is supposed to handle has several solutions. In this case we create specialised classes for each solution, and the main class then delegates the task to instances of some of these subclasses. The main parts are:

* The main class (or context)
* An abstract strategy class
* Subclasses of the strategy class implementing different solutions
* A pointer to a subclass instance in the context representing the (currently) used solution. The relevant operation in the context get forwarded to this strategy.

**\*END\***