National University of Computer and Emerging Sciences

Islamabad Campus

Object Oriented Programming (CS1004)

Course Instructor(s):
Ms. Momina Behzad
Section(s): SE-A

Final Exam

Total Time (Hrs): 3
Total Marks: 175
Total Questions: 6

Date: Jan 3, 2026

Model Solution

Roll No **Course Section**

Student Signature

Do not write below this line.

Attempt all the questions.

Exam Instructions

1. **Read all questions and instructions carefully before attempting the paper.**
2. **Question 01 must be attempted on the answer sheet.**
   All remaining questions must be attempted on the question paper.
3. **Use the provided bubble sheet to mark answers for MCQs only.**
4. **Overwriting, cutting, or use of correction fluid is not allowed.**
5. **Answers must be neat, clear, and to the point.**
   Untidy or unclear answers will not be awarded marks.
6. **Ensure that the question paper contains 6 questions in total and 23 pages before starting the exam.**
   Any discrepancy must be reported immediately to the invigilator.

[CLO 1: Demonstrate the basic concepts of object-oriented programming]

Q1: Select the correct answer from the following options and fill in the bubble sheet at the end. Marks will only be awarded according to bubble sheet. Overwriting and cutting will deduct the marks. **[50 marks]**

1: What happens in the code below?

```
int x = 10;
int *p = &x;
cout << *p++ << " " << *p;
```

a) Prints 10 11

b) Prints 10 then garbage

c) Compile-time error

d) Undefined behavior

*p++ increment the pointer not the value so it is refrencing to some garbage value.

2: Select the output of the following code snippet.

```
int arr[] = {1,2,3,4};
int *p = arr;
cout << *(p + 3);
```

p → arr[0]
*(p+3) → arr[3] = 4.

a) 3
b) 4
c) Address of arr[3]
d) Runtime error

3: Which statement is true for the following code?

```
int x = 5;
int *const p = &x;
*p = 10;
```

*const p means constant pointer not constant value so you can modify the value.

a) Compile-time error
b) x becomes 10
c) p can be reassigned
d) Undefined behavior

4: Which statement is true for the following code?

```
int x = 10;
const int *p = &x;
*p = 20;
```

const int *p => means that you cannot modify the value using p.

a) x becomes 20
b) Warning only
c) Compile-time error
d) Runtime error

5: What is the output for the following code?

```
int a = 5;
void f(int *p){
   *p = *p + 1;
}
f(&a);
```

```
cout << a;
```

a) 5

b) 6 → as a is passed by reference.

c) Garbage

d) Error

6: What does sizeof(pointer) generally return?

a) Size of pointed data

b) Size of array

c) Size of memory address

d) Depends on data type

A pointer stores an address so size of will get the address of pointer for you

7: Output?

```
int *p;
cout << *p;
```

a) 0

b) Garbage value

c) Compile-time error

d) Undefined behavior

Uninitialized pointer.

8: Output?

```
int f(int n){
  if(n==0) return 1;
  return n * f(n-1);
}
cout << f(0);
```

a) 0

b) 1 → Base case trigger.

c) Stack overflow

d) Garbage

9: Which condition causes infinite recursion?

a) Missing base case

b) Base case unreachable

c) Recursive call not changing parameters

d) All of the above

10: Output?

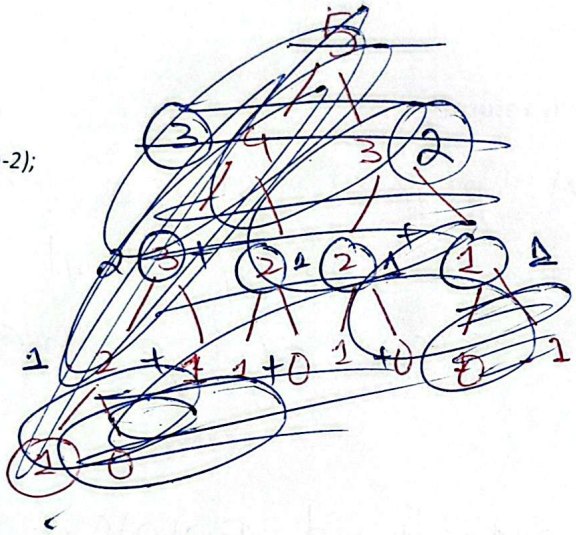
```
int f(int n){
  if(n<=1) return n;
  return f(n-1) + f(n-2);
}
cout << f(5);
```

a) 5

b) 8

c) 7

d) 6



11: Recursion always consumes more memory than iteration because:

a) Extra variables

b) Stack frames

c) Heap usage

d) Compiler choice

Each recusive call adds a new stack.

12: What is correct about this code?

```
void f(){
  static int x = 0;
  x++;
  cout << x << " ";
  f();
}
```

a) Prints infinite 1s

b) Prints increasing numbers until stack overflow → because recursion never stops.

c) Causes compile error

d) Resets x every call

13: Which concept is achieved using virtual functions?

a) Encapsulation

b) Inheritance

c) Polymorphism

d) Abstraction

14: Output?

```
class A{
public:
   A(){ cout<<"A"; }
};
class B: public A{
public:
   B(){ cout<<"B"; }
};
int main(){
   B obj;
}
```

a) B

b) A

c) AB ⟶ Base constructor runs first.

d) BA

15: If a base class destructor is not virtual and object is deleted via base pointer:

a) Safe

b) Derived destructor not called

c) Compile-time error

d) Destructor called twice

Deleting via base pointer without virtual destructor causes incomplete destruction.

16: Which cannot be inherited?

a) Public members

b) Protected members

c) Private members

d) Constructors

17: What is true about the following code?

```
class A{
   int x;
};
int main(){
   A obj;
   cout << sizeof(obj);
}
```

a) Size is 0

b) Size is >= 1 ⟶ empty object still need unique address for class member variable.

c) Compile-time error

d) Depends on compiler only

18: Which violates encapsulation?

a) Getters

b) Setters

c) Public data members

d) Private functions

19: Which operator cannot be overloaded?

a) +

b) []

c) :: scope resolution cannot be overloaded

d) <<

20: What does this overload support?

```
class A{
public:
   int x;
   A(int x):x(x){}
   A operator+(A obj){
      return A(x + obj.x);
   }
};
```

a) A + A → as return + passing both are objects of class A.
b) int + A
c) A + int
d) All

21: Overloading << usually requires:

a) Member function
b) Friend function << needs operand as ostream.
c) Static function
d) Virtual function

22: A operator++(int); This represents:

a) Prefix ++
b) Postfix ++ int dummy parameter is for postfix.
c) Both
d) Invalid

23: Operator overloading changes:

a) Operator precedence
b) Operator associativity
c) Operator meaning for user-defined types
d) Compiler behavior

24: Why = operator is often overloaded carefully?

a) Memory leaks

b) Shallow copy issues

c) Syntax errors

d) Slower execution

assignment operator can cause shallow copy.

25: Friend functions:

a) Are class members

b) Can access private members

c) Inherit class

d) Are virtual

26: Which is true?

```
class A{
   int x;
   friend void f(A);
};
```

a) f is member of A

b) f can access x

c) f must be static

d) f must be defined inside class

friendship grant access not membership

27: Output?

```
void f(){
   static int x = 0;
   x++;
   cout << x << " ";
}
int main(){
   f(); f(); f();
}
```

a) 1 1 1

b) 1 2 3

c) 0 1 2

d) Garbage

28: Static member variables are:

a) Per object

b) Per class

c) Per function call

d) Per scope

29: Static member functions:

a) Can access non-static members directly

b) Need object to be called

c) Belong to class, not objects

d) Are virtual by default

30: Where must x be defined?

```
class A{
   static int x;
};
```

a) Inside class only

b) Inside constructor

c) Outside class

d) No definition needed

Static members need global scope.

31: Lifetime of static local variable is:

a) Until block ends

b) Until function ends

c) Entire program

d) Until object is destroyed

32: Output?

```
int f(int n){
   static int x = 0;
   if(n==0) return x;
   x++;
   return f(n-1);
}
cout << f(3);
```

a) 0

b) 1

c) 3

d) Undefined

33: Which combination enables runtime polymorphism?

a) Templates + inline

b) Virtual functions + base pointer

c) Static binding

d) Friend functions

34: int *p = new int[5]; delete p; This causes?:

a) Memory leak

b) Undefined behavior

c) Compile error

d) Correct deletion

new [] must be deleted with delete []

35: Which feature breaks pure OOP principles in C++?

a) Templates

b) Multiple inheritance

c) Friend functions

d) Virtual functions

36: Output?

```
int x = 10;
int &r = x;
r++;
cout << x;
```

a) 10

b) 11

c) Garbage

d) Error

37: Destructor is called when:

a) Object created

b) Scope ends

c) Pointer declared

d) Function called

38: Which constructor is used in the following code?

```
class A{
   int x;
public:
   A(int x):x(x){}
};
A a1 = 5;
```

→ it invoke constructor with parameter.

a) Default

b) Copy

c) Parameterized

d) None

39: Which is illegal?

a) Static constructor

b) Static data member

c) Static member function

d) Static local variable

40: Which f() is called?

```
class A{
public:
   virtual void f(){}
};
class B: public A{
public:
   void f(){}
};
A* p = new B();
p->f();
```

a) A::f

b) B::f

c) Compile error

d) Ambiguous

41: Which situation forces runtime polymorphism instead of compile-time binding?

a) Function overloading

b) Templates

c) Base class pointer referring to derived object with virtual functions

d) Inline functions

42: Why are base class destructors usually declared virtual?

a) To improve performance

b) To allow constructor chaining

c) To ensure complete object destruction via base pointer

d) To support multiple inheritance

43: Which statement about object slicing is correct?

a) It occurs only with pointers

b) It preserves derived class behavior

c) It happens when a derived object is assigned to a base object

d) It is prevented by virtual functions

44: In multiple inheritance, the diamond problem primarily causes:

a) Increased memory usage

b) Ambiguity in base class member access

c) Slower runtime binding

d) Constructor overloading conflicts

45: Which feature of OOP is most compromised by excessive use of friend functions?

a) Polymorphism

b) Inheritance

c) Encapsulation

d) Abstraction

46: Why can constructors not be virtual in C++?

a) Constructors do not return values

b) Virtual table is not fully constructed

c) Compiler limitation

d) Constructors are inline

47: Which statement about abstraction vs encapsulation is correct?

a) Both hide implementation details

b) Encapsulation hides data, abstraction hides behavior

c) Abstraction focuses on what, encapsulation on how

d) They are interchangeable concepts

48: Which condition is mandatory for achieving runtime polymorphism in C++?

a) Function overloading

b) Inheritance only

c) Virtual functions accessed through base class pointer or reference

d) Templates with specialization

49: Why does polymorphic behavior not work when objects are passed by value?

a) Virtual functions are disabled

b) Copy constructor is invoked

c) Object slicing removes derived part

d) Reference binding fails

50: Which statement best distinguishes compile-time polymorphism from runtime polymorphism?

a) Compile-time polymorphism is slower

b) Runtime polymorphism requires inheritance and dynamic binding

c) Compile-time polymorphism uses virtual tables

d) Runtime polymorphism works without base classes

Bonus Question [5 Marks]

Question	Answer/Justification
A class Student behaves: • innocent in class • silent in exam • confident in viva Which OOP concept explains this personality switch?	Polymorphism

(5 Bonus marks for attempting it even if the answer is wrong. zero marks for not attempting it).

National University of Computer and Emerging Sciences

Islamabad Campus

Answer Sheet

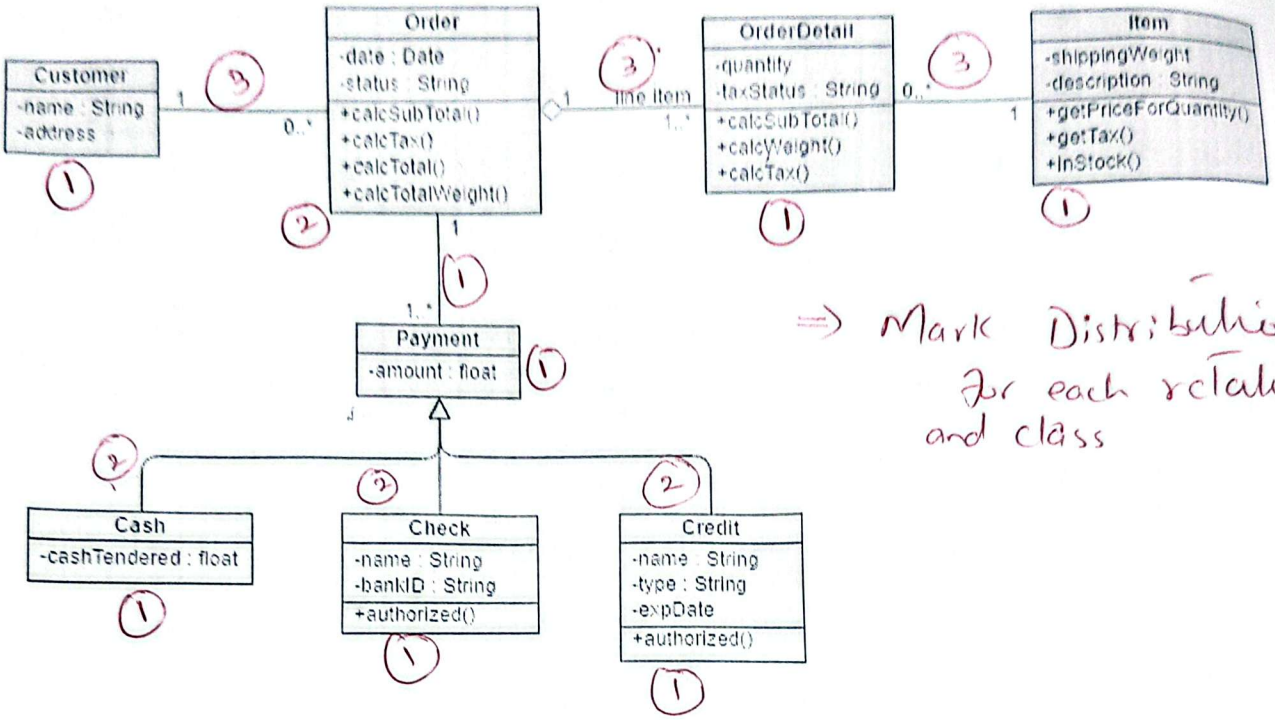
Name:

Date:

1.	A	B	C	**D**	21.	A	**B**	C	D
2.	A	**B**	C	D	22.	A	**B**	C	D
3.	A	**B**	C	D	23.	A	B	**C**	D
4.	A	B	**C**	D	24.	A	**B**	C	D
5.	A	**B**	C	D	25.	A	**B**	C	D
6.	A	B	**C**	D	26.	A	**B**	C	D
7.	A	B	C	**D**	27.	A	**B**	C	D
8.	A	**B**	C	D	28.	A	**B**	C	D
9.	A	B	C	**D**	29.	A	B	**D**	D
10.	A	**B**	C	D	30.	A	B	**C**	D
11.	A	**B**	C	D	31.	A	B	**D**	D
12.	A	**B**	C	D	32.	A	B	**C**	D
13.	A	B	**C**	D	33.	A	**B**	C	D
14.	A	B	**C**	D	34.	A	**B**	C	D
15.	A	**B**	C	D	35.	A	B	**D**	D
16.	A	B	C	**D**	36.	A	**B**	C	D
17.	A	**B**	C	D	37.	A	**B**	C	D
18.	A	B	**C**	D	38.	A	B	**D**	D
19.	A	B	**C**	D	39.	**A**	B	C	D
20.	**A**	B	C	D	40.	**A**	**B**	C	D

[CLO 4: Apply good programming practices]

Q2: Implement the following class diagram. Try to apply the concepts of OOP. Write neat and clean code that clearly identifies all data members, functions, relations between classes. Attempt it on Answer Sheet. **[25 marks]**



⇒ Mark Distribution for each relation and class

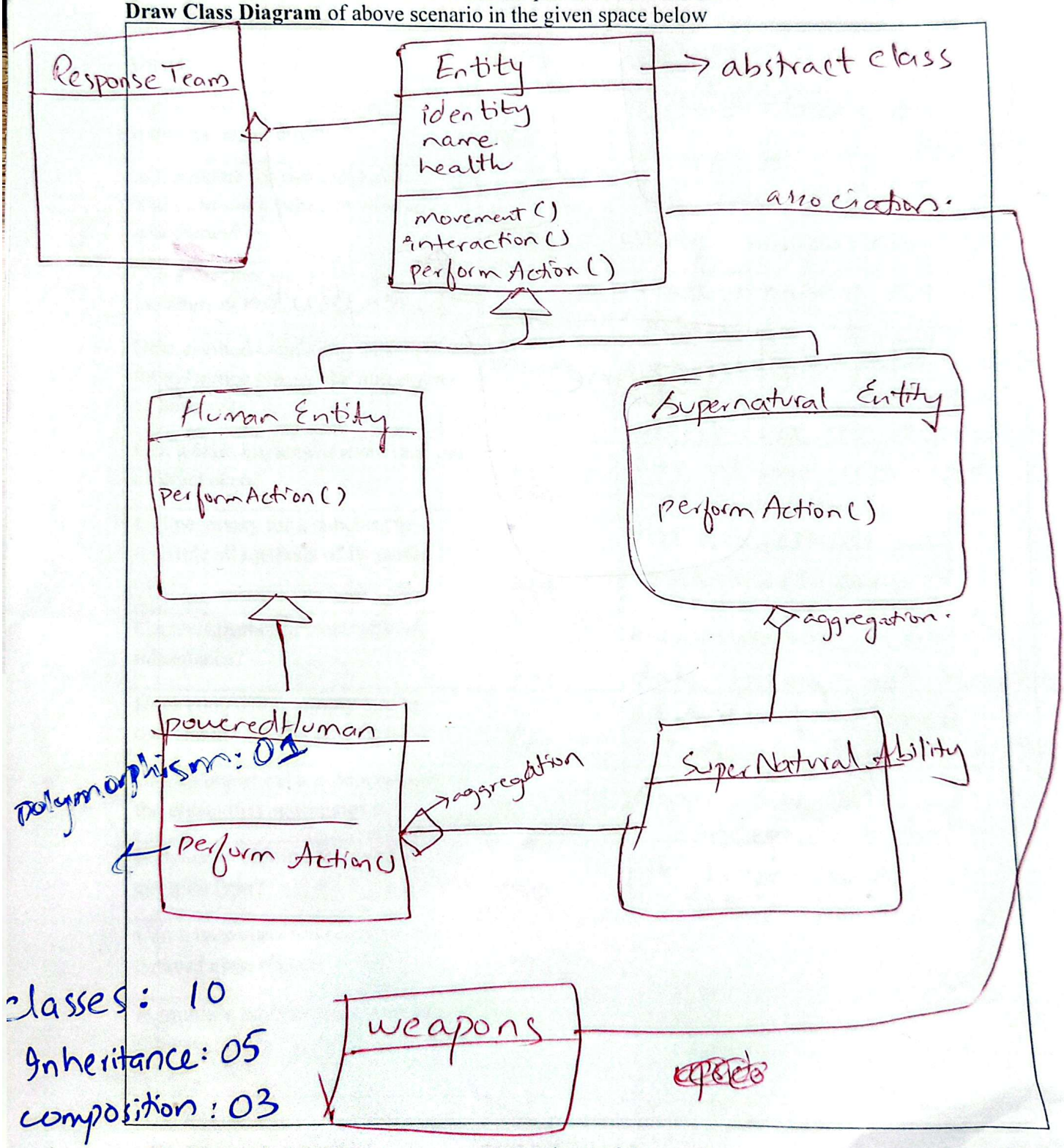
[CLO 3: Model an algorithmic solution for a given problem using OOP.]

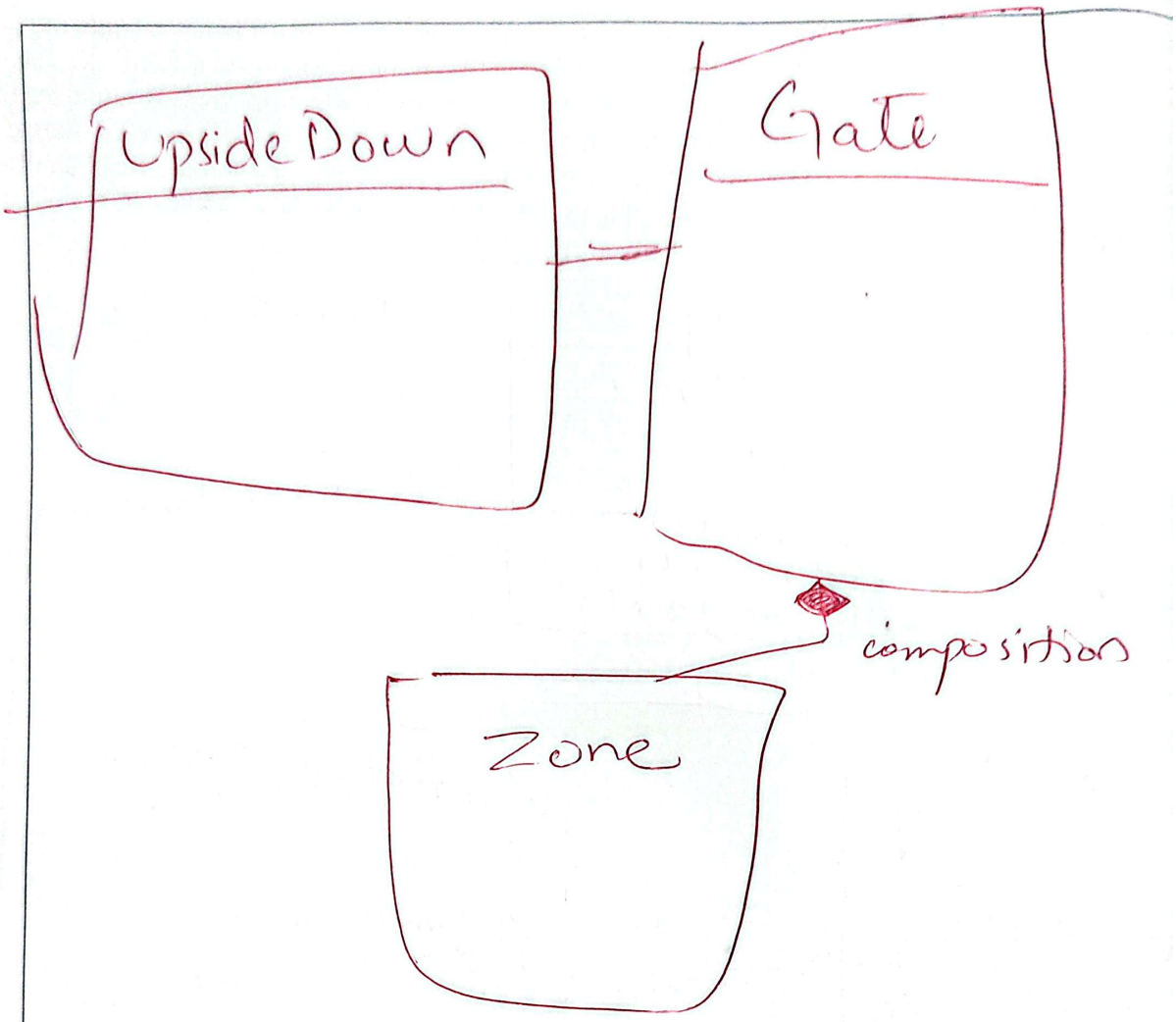
Q3: **[25 marks]**

The Hawkins National Laboratory develops a software system called the Supernatural Incident Management System to monitor and respond to threats originating from the Upside Down, using object-oriented programming principles to model real-world entities and behaviors. At the core of the system is an abstract class named Entity, which defines common attributes such as identity, name, and health level, along with general behaviors like movement and interaction, while leaving critical actions to be implemented by subclasses. This abstract class is inherited by specialized classes such as Human Entity and Supernatural Entity, which further give rise to concrete classes including powered humans like Eleven, Will and supernatural creatures such as the Demogorgon and the Mind Flayer, demonstrating inheritance and hierarchical classification. Encapsulation is enforced by keeping sensitive data, such as health and energy levels, private and allowing controlled access through public methods that apply damage, restore health, or consume energy, ensuring data integrity and system security. Polymorphism is achieved through method overriding, where a common method such as performAction() is implemented differently by each subclass, allowing the system to invoke the appropriate behavior at runtime based on the actual object type rather than the reference type. The system also defines an abstraction named Supernatural Ability, which declares behaviors related to activating powers and energy consumption, and is implemented by classes like Eleven, Will and the Mind Flayer to enable shared capabilities without enforcing a rigid inheritance structure. Association is represented through the relationship between entities and weapons, where characters may use weapons such as bats or firearms without owning their lifecycle, while aggregation is modeled through a Response Team class

that groups multiple entities for coordinated missions, allowing members to exist independently of the team. Composition is illustrated through the relationship between the Upside Down Zone and Gate objects, where gates cannot exist without the zone and are destroyed when the zone is eliminated, reflecting strong ownership. Dynamic binding ensures that when actions are triggered during simulations or encounters, the correct subclass implementations are executed at runtime, enabling the system to remain extensible, reusable, and adaptable as new characters or threats emerge.

Draw Class Diagram of above scenario in the given space below





CLO 2: Apply OOP concepts(Encapsulation, Inheritance, Polymorphism, Abstraction) to computing problems for the related program]

Q4: Answer the following questions based by applying object-oriented programming concepts. For each statement, indicate Yes or No, and provide a brief justification for your answer. **[40 marks]**

Question	Yes / No	Justification
Can an abstract class be used to create an object directly?	No	Abstract classes are incomplete by Design.
Is it possible for two unrelated classes to share behavior without inheritance?	Yes	They can share behavior using composition, associat etc
Can a subclass access private data members of its parent class directly?	No	Private members are only accessible to declaring class
Does method overriding depend on the reference type or the object type at runtime?	Object type	Runtime polymorphism resolves calls based on the actual object, not the reference.
Can a class implement more than one abstract class?	Yes.	c++ allows multiple inheritance.
Is it necessary for a subclass to override all methods of its parent class?	No	not all methods only abstract ones.
Can encapsulation exist without inheritance?	Yes	encapsulation is about data hiding, not hirerchy.
Does composition imply strong ownership between two classes?	Yes	lifetime of contained object depends on the owner
Can an object exist independently of the object that aggregates it?	Yes	Aggregation allows independent lifetime.
Is dynamic binding resolved at compile time?	No	it is resolved at runtime.
Can a base class reference point to a derived class object?	Yes	basic polymorphism
Is multiple inheritance of classes allowed in all object-oriented languages?	No	not allowed in languages like java etc due to diamond problem.

Can polymorphism occur without method overriding?	Yes	compile time polymorphism occurs via overloading.
Does increasing inheritance depth always improve code reuse?	No	it makes it complicated.
Can abstract classes contain method implementations?	Yes.	except pure virtual functions all methods can have implementa
Is it possible to change program behavior at runtime without modifying existing code?	Yes	achieved using polymorphism →
Can two classes work together without one owning the other?	Yes	In association.
Does abstraction require hiding all implementation details?	No	constructor is not inherited so it can be ove
Can a constructor be overridden in a subclass?	No	hide irrelevant details not everything.
Can a class be both abstract and partially implemented?	Yes	a class can have abstract & non abstract method

[CLO 3: Model an algorithmic solution for a given problem using OOP.]

Q5: From the given poem, **identify** class relationships present in the system. For each relationship name the two classes involved and Specify the type of relationship (Composition, Aggregation Inheritance, or Association). Write the line of poem that indicated the relationship in justification section. Fill out the table below: **[10 marks**

The Clockwork Theatre

In a Clockwork Theatre, a Stage stands still,
It owns no actors, yet shapes every skill.
Scenes are prepared before curtains rise,
A silent framework where action applies.
An Actor enters with a learned role,
Each trained from a common dramatic soul.
Some speak in whispers, some shout their lines,
Different styles, same inherited signs.
When the Script calls for a moment to speak,
All actors respond, though the voices are unique.
The same cue sent, the same method called,
Yet emotions differ when the words fall.
An Actor carries a Costume each night,

Stitched to the role, removed from sight.
Without the Actor, the Costume stays none,
Their existence ends when the role is done.
A Director guides but does not stay,
Moving between stages, night by day.
The Director shapes but does not belong,
Leaving the Stage when the show moves on.
When the Theatre closes, the Stage remains,
Actors disperse, trained but free from chains.
The play may end, the design survives,
Ready to host new dramatic lives.

Class A	Class B	Type of Relationship	Justification
Theatre	Stage	composition	stage exist as part of theater, "a stage stands still".
Stage	Actor	Association	"An actor enters with a leared "gle"
Actor	costume	composition	without the actor costume stays none.....
Actor	Role	Inheritance. association.	Each trained from a comon soul. some speak in whisper some shout....
Director	Stage	Association	a director guides but does not stay....
Director	Theater	Association	The director shapes but does not belong
Theater	Design	Agregation	The play may ends.. The design survive
Actor	Script	Association	when the script calls for a
Actor	Performance	Aggregation	when the theater closes... actor dispense...
Theatre	Actor	Association	Ready to host new dramatic lives...

[CLO 2: Model an algorithmic solution for a given problem using OOP.]

Q6: Trace the following code snippets. If there is an error identify the error and write its corrected version. If there is no error than write the output of the given code. Each Output or corrected version carry exactly 5 marks. **[25 marks]**

Code	Corrected Version or Output

Part A:
class Base {
public:
void show() { cout << "Base\n"; }
};
↓
bcz show is not virtual.
	Base.

```
class Derived : public Base {
public:
   void show() { cout << "Derived\n"; }
};

int main() {
   Base *b = new Derived();
   b->show();
   return 0;
}
```

Part B:

```
class Shape {
public:
   virtual void draw() { cout << "Shape\n"; }
};

class Circle : public Shape {
public:
   void draw() { cout << "Circle\n"; }
};

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

draw is virtual

Circle.

Part C:

```
class Test {
public:
   static int count;
   Test() { count++; }
};

int Test::count = 0;

int main() {
   Test t1, t2;
   cout << Test::count;
   return 0;
}
```

02

Part D:

```
class Parent {
protected:
   int x;
};

class Child : private Parent {
public:
   void setX(int val) { x = val; }
```

(1) make it public

error:- x is protected in parent & parent is privately inherited. so x is not accessible outside child.

```
    int getX() { return x; }
};

int main() {
    Child c;
    c.setX(5);
    cout << c.x;
    return 0;
}
```

Anything from (1) or (2) works.

(2) → cout << c.getX()

Part E:

```
class Alpha {
public:
    virtual void f() { cout << "Alpha"; }
};

class Beta : public Alpha {
public:
    void f(int x) { cout << x; }
};

int main() {
    Beta b;
    Alpha *a = &b;
    a->f();
    return 0;
}
```

Alpha.

→ Beta::f(int) does not override Alpha::f() it hides it instead due to different signature.

→ not overriding, it works as overloading.