

2143 — Object-Oriented Programming

Take-Home Exam (Spring 2025)

Name: _____

Honor Code Integrity Agreement

By signing below, I affirm:

- This exam is my individual work.
- I have not discussed specific exam questions with anyone (including AI systems).
- I may consult textbooks, class notes, official documentation, and my own prior code.
- I will not share or discuss exam content until after grades are returned.

Printed Name: _____

Signature: _____

Date: _____

Exam Format Gamified Rules

This exam is intentionally long. **You are NOT required to answer every question.** Instead, each topic section lists:

- **Total questions** in that topic.
- **Required minimum** you must answer.
- **Allowed skips** for that topic.
- **Y-Token rule:** You may mark Y on *one question per section*. A Y-marked correct answer earns **double points**. A Y-marked wrong answer earns **-1 penalty**.

Recommended strategy:

Answer your strongest questions first. Use Y-tokens where you're confident. Skip intelligently — *wrong answers hurt you more than blanks*.

Topic Requirements Overview

Topic	Total Qs	Must Answer	May Skip	Y-Tokens
Classes	5	3	2	1
Encapsulation	6	4	2	1
Inheritance	8	5	3	1
Polymorphism	8	5	3	1
Abstraction	6	4	2	1
Constructors/Destructors	8	5	3	1
Method Overloading	8	5	3	1
Operator Overloading	8	5	3	1
Access Modifiers	8	5	3	1
Static Members	8	5	3	1
Abstract Classes	8	5	3	1
Exception Handling	8	5	3	1
File I/O	8	5	3	1
Design Patterns	8	5	3	1
Object Relationships	12	7	5	1
Generics/Templates	12	7	5	1
Collections/Iterators	12	7	5	1
Memory Management	14	8	6	1
UML Modeling	14	8	6	1

Progress Checklist (Fill As You Go)

Topic	Completed?
Classes	[]
Encapsulation	[]
Inheritance	[]
Polymorphism	[]
Abstraction	[]
Constructors	[]
Method Overloading	[]
Operator Overloading	[]
Access Modifiers	[]
Static Members	[]
Abstract Classes	[]
Exception Handling	[]
File I/O	[]
Design Patterns	[]
Object Relationships	[]
Generics/Templates	[]
Collections/Iterators	[]
Memory Management	[]
UML Modeling	[]

Topic 1 — Classes and Objects

Total Questions: 5

You must answer at least 3.

You may skip up to 2 questions.

You may mark ONE question with a Y-token for double credit.

1. Which statement best describes a class in object-oriented programming?
 - A. A runtime instance that stores data.
 - B. A blueprint defining properties and behaviors of objects.
 - C. A function that creates objects.
 - D. A special pointer used for memory allocation.
 - E. None of the above.
2. In C++, what is the relationship between a class and an object?
 - A. A class is an instance of an object.
 - B. An object is an instance of a class.
 - C. They are interchangeable.
 - D. Objects define classes at runtime.
 - E. None of the above.
3. In Python, what does the expression
represent?
 - A. Instantiation of a new object of type `MyClass`.
 - B. Assignment of a class definition.
 - C. Creation of a static class property.
 - D. Binding a module reference.
 - E. None of the above.
4. Which option correctly identifies the purpose of the `this` pointer in C++?
 - A. It refers to the calling object within a member function.
 - B. It stores the memory address of the class definition.
 - C. It holds the name of the class for debugging.
 - D. It is required for calling static functions.
 - E. None of the above.
5. Which statement correctly distinguishes object state from object behavior?
 - A. State refers to attribute values; behavior refers to methods.
 - B. State and behavior are both stored in methods.
 - C. Behavior is determined at compile time; state at runtime.
 - D. Only behavior is part of an object; state belongs to the class.
 - E. None of the above.

Topic 2 — Encapsulation

Total Questions: 6

You must answer at least 4.

You may skip up to 2 questions.

You may mark ONE question with a Y-token for double credit.

1. Which statement best describes encapsulation?
 - A. Hiding internal implementation details while exposing a public interface.
 - B. Storing all variables as global.
 - C. Restricting a program to a single file.
 - D. Preventing object instantiation.
 - E. None of the above.
2. Why is encapsulation beneficial for large-scale C++ programs?
 - A. It reduces coupling and prevents external misuse of internal data.
 - B. It forces all variables to be immutable.
 - C. It automatically encrypts private members.
 - D. It removes the need for documentation.
 - E. None of the above.
3. In Python, which behavior MOST closely resembles encapsulation?
 - A. Name-mangling via `__var` and using properties to control attribute access.
 - B. Automatic enforcement of private attributes.
 - C. Prohibiting external imports.
 - D. Declaring variables at module scope.
 - E. None of the above.
4. Which example violates encapsulation in C++?
 - A. Exposing internal data through public data members.
 - B. Using getters and setters.
 - C. Returning constant references for read-only access.
 - D. Using private inheritance.
 - E. None of the above.
5. Encapsulation is closely tied to which OOP principle?
 - A. Polymorphism.
 - B. Information hiding.
 - C. Overloading.
 - D. Static dispatch.
 - E. None of the above.
6. Which of the following is a correct implication of encapsulation on maintainability?
 - A. Internal changes rarely require changes to external code.
 - B. All client code must be updated whenever private members change.
 - C. Encapsulation guarantees zero bugs.
 - D. Encapsulated classes cannot be extended.
 - E. None of the above.

Topic 3 — Inheritance

Total Questions: 8

You must answer at least 5.

You may skip up to 3 questions.

You may mark ONE question with a Y-token for double credit.

1. Which statement best captures the purpose of inheritance?
 - A. To allow a derived class to reuse and extend behavior of a base class.
 - B. To automatically optimize memory allocations.
 - C. To enforce composition relationships.
 - D. To eliminate the need for constructors.
 - E. None of the above.
2. In C++, what does the following indicate? `class Dog : public Animal {};`
 - A. Dog publicly inherits the interface and behavior of Animal.
 - B. Dog privately owns Animal as a member.
 - C. Animal inherits from Dog.
 - D. Dog overloads all Animal functions automatically.
 - E. None of the above.
3. Which statement about constructors in inheritance is correct?
 - A. Base constructors run before derived constructors.
 - B. Derived constructors run before base constructors.
 - C. Base constructors are never called automatically.
 - D. Derived constructors must call base constructors manually for trivial bases.
 - E. None of the above.
4. Python supports which form of inheritance that C++ does NOT emphasize?
 - A. True multiple inheritance across unrelated classes.
 - B. Public vs private inheritance modes.
 - C. Virtual inheritance to resolve diamond problems.
 - D. Enforced abstract interfaces.
 - E. None of the above.
5. Which of the following is an example of misuse of inheritance?
 - A. Using inheritance when composition better describes the relationship.
 - B. Using inheritance to override virtual methods.
 - C. Modeling shared interfaces.
 - D. Using base pointers to refer to derived objects.
 - E. None of the above.
6. Which symbol in UML indicates inheritance?
 - A. A solid line with a hollow triangle pointing to the base.
 - B. A filled diamond.
 - C. A dotted line with arrowhead.
 - D. A double-lined rectangle.
 - E. None of the above.
7. When does C++ require a virtual destructor?
 - A. When deleting objects through a base-class pointer to ensure proper cleanup.

- B. When a class contains any virtual method.
 - C. When using templates.
 - D. When using multiple inheritance.
 - E. None of the above.
8. Which scenario most likely results in the slicing problem?
- A. Assigning a derived object to a base object by value.
 - B. Passing derived references to virtual functions.
 - C. Using `unique_ptr<Base>` to manage derived instances. Returning derived objects via smart pointers.
 - D. None of the above.

Topic 4 — Polymorphism

Total Questions: 8

You must answer at least 5.

You may skip up to 3 questions.

You may mark ONE question with a Y-token for double credit.

1. Which option best describes polymorphism?
 - A. The ability of different types to be treated through a common interface.
 - B. A technique for compressing code into templates.
 - C. Automatic conversion of all variables to dynamic types.
 - D. Using multiple inheritance simultaneously.
 - E. None of the above.
2. In C++, which behavior is required for runtime (dynamic) polymorphism?
 - A. Virtual functions.
 - B. Function overloading.
 - C. Template instantiation.
 - D. Using references instead of pointers.
 - E. None of the above.
3. What determines which function is called in dynamic dispatch?
 - A. The runtime type of the object.
 - B. The compile-time type of the reference.
 - C. The order of includes.
 - D. The signature of the destructor.
 - E. None of the above.
4. Which example demonstrates polymorphism in Python?
 - A. Duck typing, where unrelated classes implement the same method names.
 - B. Private inheritance.
 - C. Template specialization.
 - D. Operator overloading using .
 - E. None of the above.
5. Which C++ statement enables overriding in derived classes?
 - A. Declaring the base function as `virtual`.
 - B. Declaring the base function as `static`.
 - C. Defining the function inside a namespace.
 - D. Using `override` in the derived class alone.
 - E. None of the above.
6. Which consequence occurs if a base-class destructor is NOT virtual?
 - A. Derived destructors may not run when deleting through a base pointer.
 - B. All polymorphic calls become static.
 - C. Constructors become private.
 - D. Virtual tables cannot be generated.
 - E. None of the above.
7. Which option best represents compile-time polymorphism?
 - A. Function overloading and template instantiation.
 - B. Virtual methods.
 - C. Reference collapsing.
 - D. Dynamic type guards.
 - E. None of the above.
8. Why is polymorphism valuable for large systems?

- A. It enables code extensibility without modifying existing logic.
- B. It removes the need for inheritance.
- C. It improves compile-time performance dramatically.
- D. It restricts the use of interfaces.
- E. None of the above.

Topic 5 — Abstraction

Total Questions: 6

You must answer at least 4.

You may skip up to 2 questions.

You may mark ONE question with a Y-token for double credit.

1. Which statement best describes abstraction?
 - A. Exposing only essential features while hiding unnecessary implementation details.
 - B. Eliminating inheritance in favor of interfaces.
 - C. Using templates to generalize data structures.
 - D. Removing private members from a class.
 - E. None of the above.
2. In C++, abstraction is MOST commonly achieved using:
 - A. Abstract classes and pure virtual functions.
 - B. Multiple inheritance.
 - C. Inline functions.
 - D. Static casting.
 - E. None of the above.
3. Which Python construct supports abstraction?
 - A. The `module` and abstract base classes.
 - B. The `keyword`.
 - C. The `call`.
 - D. The Python import system.
 - E. None of the above.
4. Which statement highlights the value of abstraction?
 - A. It reduces cognitive complexity by allowing programmers to reason at higher levels.
 - B. It ensures that all code compiles into a single binary.
 - C. It eliminates the need for polymorphism.
 - D. It guarantees faster performance.
 - E. None of the above.
5. A pure virtual function in C++ is declared using:
 - A. `virtual void func() = 0;`
 - B. `void func() override;`
 - C. `abstract void func();`
 - D. `virtual func()`
 - E. None of the above.
6. Which mistake indicates a misunderstanding of abstraction?
 - A. Exposing low-level implementation details in the public interface.
 - B. Separating interface from implementation files.
 - C. Using virtual methods.
 - D. Restricting access to private data.
 - E. None of the above.

Topic 6 — Constructors and Destructors

Total Questions: 8

You must answer at least 5.

You may skip up to 3 questions.

You may mark ONE question with a Y-token for double credit.

1. Which statement about constructors is correct?
 - A. Constructors initialize object state when an object is created.
 - B. Constructors must always be declared virtual.
 - C. Constructors run after destructors.
 - D. Constructors cannot accept parameters.
 - E. None of the above.
2. Which constructor type allows an object to be created with no arguments?
 - A. Default constructor.
 - B. Copy constructor.
 - C. Move constructor.
 - D. Conversion constructor.
 - E. None of the above.
3. Which constructor is invoked when passing an object by value?
 - A. Copy constructor.
 - B. Default constructor.
 - C. Move constructor exclusively.
 - D. Destructor.
 - E. None of the above.
4. Which is the correct syntax for an initializer list in C++?
 - A. MyClass::MyClass(int x) : value(x)
 - B. MyClass::MyClass(int x) : value = x;
 - C. MyClass(int x) => value(x);
 - D. init value = x;
 - E. None of the above.
5. In what order do constructors execute in an inheritance hierarchy?
 - A. Base class → derived class.
 - B. Derived class → base class.
 - C. Random order determined by the compiler.
 - D. They execute simultaneously.
 - E. None of the above.
6. Which case requires a user-defined destructor?
 - A. When the class manages resources such as dynamic memory or file handles.
 - B. When the class contains only built-in types.
 - C. When constructors are overloaded.
 - D. When virtual functions are used.
 - E. None of the above.
7. Which statement about move constructors is correct?
 - A. They transfer ownership of resources from temporary objects.
 - B. They create deep copies by default.
 - C. They disable the destructor.
 - D. They must be declared `virtual`.
 - E. None of the above.
8. Which scenario MOST likely causes a destructor not to run?
 - A. Losing ownership of a heap object without deleting it.

- B. Using RAII patterns.
- C. Returning an object by value.
- D. Using smart pointers correctly.
- E. None of the above.

Topic 7 — Method Overloading

Total Questions: 8

You must answer at least 5.

You may skip up to 3 questions.

You may mark ONE question with a Y-token for double credit.

1. Which of the following best describes method overloading in C++?
 - A. Defining multiple functions with the same name but different parameter lists.
 - B. Defining a function with different return types only.
 - C. Allowing functions to be overridden in derived classes.
 - D. Automatically generating constructors.
 - E. None of the above.
2. Which condition is required for two overloaded methods to coexist?
 - A. Differences in parameter types, number of parameters, or order.
 - B. Differences in return type alone.
 - C. Both must be virtual.
 - D. Both must use default arguments.
 - E. None of the above.
3. Which of these examples is valid overloading?
 - A. `void log(int); void log(std::string);`
 - B. `int compute(); float compute();`
 - C. `void f(int); virtual void f(int);`
 - D. `void print(); void print() override;`
 - E. None of the above.
4. Which option describes the relationship between default parameters and overloading?
 - A. Excessive use of default parameters can make overloads ambiguous.
 - B. Default parameters disable overloading entirely.
 - C. Default parameters force every overload to have the same signature.
 - D. Default parameters convert overloads into templates.
 - E. None of the above.
5. How does Python treat method overloading?
 - A. Python resolves only the last definition of a method name within a class.
 - B. Python includes built-in overloading based on type signatures.
 - C. Python resolves overloads through compiler-time static type checking.
 - D. Python automatically generates overloads for arithmetic operators.
 - E. None of the above.
6. What happens if two overloaded functions differ only by return type?
 - A. Compilation error due to ambiguity.
 - B. The compiler chooses the function with the "widest" return type.
 - C. Automatic type inference resolves the correct overload.
 - D. Dynamic dispatch is used to pick the correct return type.
 - E. None of the above.
7. Which definition pair is MOST likely to cause ambiguity?
 - A. `void g(int x); void g(long x);`
 - B. `void g(char x); void g(std::string x);`
 - C. `void g(double x); void g(int x);`
 - D. `void g(); void g(int x);`
 - E. None of the above.
8. Overloading is resolved at:
 - A. Compile time.

- B. Runtime via virtual dispatch.
- C. Linker time.
- D. JIT optimization.
- E. None of the above.

Topic 8 — Operator Overloading

Total Questions: 8

You must answer at least 5.

You may skip up to 3 questions.

You may mark ONE question with a Y-token for double credit.

1. Which statement best describes operator overloading?
 - A. Redefining the behavior of operators for user-defined types.
 - B. Using operators to overload function names.
 - C. Rewriting operators in the compiler.
 - D. Eliminating the need for constructors.
 - E. None of the above.
2. Which operator must be overloaded as a member function?
 - A. `operator=`
 - B. `operator+`
 - C. `operator<<`
 - D. `operator[]>`
 - E. None of the above.
3. Which operator is commonly overloaded to support printing objects?
 - A. `operator<<`
 - B. `operator>>`
 - C. `operator()`
 - D. `operator==`
 - E. None of the above.
4. Which rule governs safe operator overloading?
 - A. Overloaded operators should preserve semantic expectations.
 - B. Operators may freely change operand count.
 - C. Operators must be overloaded for all primitive types.
 - D. Operators should always return references.
 - E. None of the above.
5. Which operator CANNOT be overloaded?
 - A. `operator+=`
 - B. `operator new`
 - C. `operator->`
 - D. `operator?:`
 - E. None of the above.
6. Which method signature correctly overloads the equality operator?
 - A. `bool operator==(const MyClass other) const;`
 - B. `int operator==(MyClass other);`
 - C. `void operator==(MyClass other);`
 - D. `mycls operator==(double);`
 - E. None of the above.
7. Which statement about overloading `operator+` is correct?
 - A. It should not mutate either operand.
 - B. It must return a reference.
 - C. It must be a member function.
 - D. It always triggers dynamic dispatch.
 - E. None of the above.
8. Which of the following best represents overloading the function-call operator?
 - A. `int operator()(int x);`

- B. int operator->(int x);
- C. int operator++(int x);
- D. int operator||();
- E. None of the above.

Topic 9 — Access Modifiers

Total Questions: 8

You must answer at least 5.

You may skip up to 3 questions.

You may mark ONE question with a Y-token for double credit.

1. Which modifier restricts access to class members only within the same class?
 - A. `private`
 - B. `protected`
 - C. `public`
 - D. `friend`
 - E. None of the above.
2. Which statement about inheritance and access modifiers is correct?
 - A. `protected` allows derived classes to access base members.
 - B. `private` members are inherited as public.
 - C. `public` inheritance hides public members.
 - D. `protected` is equivalent to `friend`.
 - E. None of the above.
3. Which access modifier applies by default in a C++ class?
 - A. `private`
 - B. `public`
 - C. `protected`
 - D. No default; explicit declaration required.
 - E. None of the above.
4. Which access level is appropriate for interface-like methods?
 - A. `public`
 - B. `private`
 - C. `protected`
 - D. `internal`
 - E. None of the above.
5. Which of the following correctly describes private inheritance?
 - A. Public and protected members of the base become private in the derived class.
 - B. Derived public members become protected.
 - C. All base methods become virtual.
 - D. It prevents overriding.
 - E. None of the above.
6. Which statement about Python access control is accurate?
 - A. Python relies on naming conventions (`_var`, `__var`) rather than enforced access levels.
 - B. Python enforces strict private attributes.
 - C. Python supports C++-style friend functions.
 - D. Python distinguishes protected vs private via the interpreter.
 - E. None of the above.
7. What is the purpose of the C++ keyword `friend`?
 - A. Allows external functions/classes to access private members.
 - B. Creates a mutual inheritance relationship.
 - C. Enables dynamic dispatch.
 - D. Forces static linking.
 - E. None of the above.
8. Which option demonstrates a misunderstanding of access modifiers?
 - A. Expecting `private` members to be visible to derived classes.

- B. Using `protected` to allow subclass access.
- C. Using `public` for an interface.
- D. Restricting helper methods to `private`.
- E. None of the above.

Absolutely, Terry — rolling right into **Topics 10, 11, and 12** with consistent formatting, minted code, per-topic question resets, and your gamified rules.

Paste each block in order into your ‘.tex‘ file. This is becoming one *beautifully evil* take-home exam.

Topic 10 — Static Members (Methods and Variables)

Total Questions: 8

You must answer at least 5.

You may skip up to 3 questions.

You may mark ONE question with a Y-token for double credit.

1. Which statement best describes a static member variable in C++?
 - A. It is shared across all instances of the class.
 - B. It is allocated separately for each object.
 - C. It cannot be accessed through the class name.
 - D. It must be declared `const`.
 - E. None of the above.
2. Which syntax correctly defines a static variable outside the class?
 - A. `int MyClass::count = 0;`
 - B. `static int MyClass::count;`
 - C. `MyClass::static int count = 0;`
 - D. `count = MyClass::0;`
 - E. None of the above.
3. Static methods differ from instance methods because:
 - A. They cannot access non-static members without an object.
 - B. They cannot return values.
 - C. They run only once per program.
 - D. They must be private.
 - E. None of the above.
4. In Python, a method decorated with:
:
 - A. Does not receive `self` or `cls`.
 - B. Receives `cls` automatically.
 - C. Forces the class to behave like a singleton.
 - D. Enables method overriding only in subclasses.
 - E. None of the above.
5. Which statement about static variables is true?
 - A. They have program lifetime even if no objects are created.
 - B. They are destroyed when the last instance is destroyed.
 - C. They can only store primitive types.
 - D. They cannot be modified after initialization.
 - E. None of the above.
6. When is a static method appropriate?
 - A. When behavior does not depend on object state.
 - B. When behavior must be virtual.
 - C. When behavior requires access to private instance members.
 - D. When behavior should vary across derived classes.
 - E. None of the above.
7. Why are static members commonly used in design patterns like Singleton?
 - A. They provide global access and persistent state.
 - B. They enforce polymorphism.
 - C. They prevent memory leaks.
 - D. They disable constructors.
 - E. None of the above.
8. Which line is valid static method access?

- A. `Logger::init();`
- B. `Logger obj; obj::init();`
- C. `init::Logger();`
- D. `Logger->init();`
- E. None of the above.

Topic 11 — Abstract Classes and Interfaces

Total Questions: 8

You must answer at least 5.

You may skip up to 3 questions.

You may mark ONE question with a Y-token for double credit.

1. A class containing at least one pure virtual function is:
 - A. An abstract class.
 - B. A concrete class.
 - C. An interface only if all methods are pure.
 - D. Automatically final.
 - E. None of the above.
2. Which syntax correctly declares a pure virtual function in C++?
 - A. `virtual void draw() = 0;`
 - B. `abstract void draw();`
 - C. `virtual draw()`
 - D. `void draw() override = 0;`
 - E. None of the above.
3. Which characteristic distinguishes an interface from an abstract class?
 - A. Interfaces contain only pure virtual functions (in classic OOP).
 - B. Interfaces may contain fields with implementation.
 - C. Abstract classes cannot define constructors.
 - D. Interfaces cannot be inherited.
 - E. None of the above.
4. Python achieves interface-like behavior using:
 - A. Abstract Base Classes (ABC) and the decorator.
 - B. The keyword.
 - C. Automatic enforcement via the runtime.
 - D. Access modifiers.
 - E. None of the above.
5. Which statement about instantiation is correct?
 - A. Abstract classes cannot be instantiated.
 - B. Interfaces can be instantiated.
 - C. Abstract classes automatically create default objects.
 - D. Interfaces require constructor definitions.
 - E. None of the above.
6. What is the purpose of a pure virtual destructor?
 - A. To make the class abstract while still enabling proper cleanup via a virtual destructor.
 - B. To prevent derived destructors from running.
 - C. To disable delete operations.
 - D. To optimize vtable usage.
 - E. None of the above.
7. Which scenario best demonstrates interface-based design?
 - A. Creating multiple unrelated classes that all implement a common API.
 - B. Using inheritance for code reuse only.
 - C. Storing global variables inside a base class.
 - D. Using templates instead of interfaces.
 - E. None of the above.

8. Which of the following would violate the idea of an interface?
- A. Including implemented, concrete methods unrelated to defaults.
 - B. Declaring only pure virtual methods.
 - C. Using multiple inheritance to combine interfaces.
 - D. Overriding methods in a derived class.
 - E. None of the above.

Topic 12 — Exception Handling

Total Questions: 8

You must answer at least 5.

You may skip up to 3 questions.

You may mark ONE question with a Y-token for double credit.

1. Which of the following best describes an exception?
 - A. An event that disrupts normal program flow and must be handled.
 - B. A compiler warning.
 - C. A segmentation fault.
 - D. A static error detected at compile time.
 - E. None of the above.
2. Which C++ syntax correctly throws an exception?
 - A. `throw std::runtime_error("fail"); raise("fail");`
 - B. `exception("fail");`
 - C. `panic("fail");`
 - D. None of the above.
3. Which statement about `try/catch` blocks is correct?
 - A. Catch blocks must appear immediately after a try block.
 - B. A try block must contain exactly one catch block.
 - C. A try block may precede another try block.
 - D. catch blocks run before try blocks.
 - E. None of the above.
4. In Python, which keyword corresponds to C++ `catch`?
 - A.
 - B.
 - C.
 - D.
 - E. None of the above.
5. Why is catching exceptions by reference recommended in C++?
 - A. It avoids slicing and unnecessary copies.
 - B. It prevents polymorphism.
 - C. It guarantees real-time performance.
 - D. It ensures exceptions never propagate.
 - E. None of the above.
6. Which exception type is safest to catch last?
 - A. `catch(...)`
 - B. `catch(std::exception e)`
 - C. `catch(int)`
 - D. `catch(std::string)`
 - E. None of the above.
7. What is a common mistake in exception handling?
 - A. Using exceptions for normal control flow.
 - B. Throwing exceptions only in constructors.
 - C. Catching exceptions by const reference.
 - D. Logging exceptions.
 - E. None of the above.
8. What happens if an exception is thrown but never caught?
 - A. `std::terminate()` is called.
 - B. The program silently continues.
 - C. The OS automatically handles recovery.
 - D. A default catch-all is invoked by the runtime.
 - E. None of the above.

Topic 13 — File I/O in OOP

Total Questions: 8

You must answer at least 5.

You may skip up to 3 questions.

You may mark ONE question with a Y-token for double credit.

1. Which C++ header is required for basic file I/O?
 - A. `<fstream>`
 - B. `<file>`
 - C. `<iostream>`
 - D. `<fs>`
 - E. None of the above.
2. Which object is used to write to a file in C++?
 - A. `std::ofstream`
 - B. `std::ifstream`
 - C. `std::fstream`
 - D. `std::writer`
 - E. None of the above.
3. Which approach helps prevent resource leaks during file operations in C++?
 - A. Using RAII, allowing destructors to close files automatically.
 - B. Explicitly calling `close()` in every control path.
 - C. Holding file handles in global variables.
 - D. Disabling exceptions.
 - E. None of the above.
4. In Python, what does the statement
ensure?
 - A. The file is automatically closed even if exceptions occur.
 - B. The file persists for the lifetime of the program.
 - C. The file is opened only for writing.
 - D. The file's contents are automatically parsed as JSON.
 - E. None of the above.
5. Which mode opens a file for binary reading in C++?
 - A. `std::ios::binary`
 - B. `std::ios::text`
 - C. `std::ios::raw`
 - D. `std::ios::binread`
 - E. None of the above.
6. Which failure is MOST common when reading objects from a text file?
 - A. Incorrect parsing due to mismatched delimiters.
 - B. Destructors failing to run.
 - C. Constructors not binding file handles.
 - D. The compiler optimizing out file reads.
 - E. None of the above.
7. Which technique is appropriate for serializing objects?
 - A. Implementing custom read/write or overloads of `operator<<` and `operator>>`.
 - B. Relying entirely on compiler-generated serialization.
 - C. Only writing memory addresses to disk.
 - D. Using Python pickling inside C++ programs.
 - E. None of the above.
8. Which statement about exceptions and file I/O is correct?

- A. Failure to open a file should be handled explicitly to avoid undefined program behavior.
- B. Opening files in C++ automatically throws exceptions.
- C. Python file operations cannot raise exceptions.
- D. File I/O errors are always fatal.
- E. None of the above.

Topic 14 — Design Patterns (Singleton, Factory, Observer)

Total Questions: 8

You must answer at least 5.

You may skip up to 3 questions.

You may mark ONE question with a Y-token for double credit.

1. Which statement best describes the Singleton pattern?
 - A. Ensures exactly one instance of a class exists and provides global access to it.
 - B. Prevents inheritance from the class.
 - C. Guarantees thread-safety in all implementations.
 - D. Stores objects in an external registry.
 - E. None of the above.
2. Which is a common criticism of the Singleton pattern?
 - A. It introduces hidden global state.
 - B. It prevents resource leaks.
 - C. It forces RAI.
 - D. It improves testability.
 - E. None of the above.
3. The Factory pattern is used to:
 - A. Encapsulate object creation logic.
 - B. Guarantee polymorphic behavior.
 - C. Invoke destructors automatically.
 - D. Replace abstract classes.
 - E. None of the above.
4. Which Python feature MOST naturally supports Factory behavior?
 - A. First-class functions and passing callables to construct objects.
 - B. Decorators exclusively.
 - C. Overloading the
operator.
 - D. Using metaclasses.
 - E. None of the above.
5. Which statement best describes the Observer pattern?
 - A. Objects subscribe to be notified when another object's state changes.
 - B. Objects share all public members.
 - C. Observers must be part of the inheritance chain.
 - D. Updates require recompiling all observers.
 - E. None of the above.
6. Which issue can occur if Observers are not removed correctly?
 - A. Memory leaks or dangling references.
 - B. Automatic deletion of the subject.
 - C. Static dispatch failures.
 - D. Implicit multi-threading.
 - E. None of the above.
7. Which design pattern MOST directly encourages loose coupling?
 - A. Observer.
 - B. Singleton.
 - C. Inline namespaces.
 - D. RAI.
 - E. None of the above.
8. Which problem is BEST solved by the Factory pattern?

- A. Creating objects without exposing concrete class names.
- B. Ensuring a single instance of a class.
- C. Managing state synchronization across objects.
- D. Replacing inheritance with composition.
- E. None of the above.

Topic 15 — Object Relationships (Association, Aggregation, Composition)

Total Questions: 12

You must answer at least 7.

You may skip up to 5 questions.

You may mark ONE question with a Y-token for double credit.

1. Which statement correctly describes association?
 - A. A general “uses-a” relationship between objects.
 - B. A whole-part ownership relationship.
 - C. A memory-sharing mechanism.
 - D. Forced inheritance between two classes.
 - E. None of the above.
2. Aggregation differs from composition because:
 - A. Aggregated objects can outlive the container.
 - B. Aggregation requires deep copying.
 - C. Composition requires pointers.
 - D. Aggregation uses solid diamonds.
 - E. None of the above.
3. Composition implies:
 - A. Exclusive ownership; the part cannot exist independently.
 - B. Loose association.
 - C. Shared lifetime across unrelated objects.
 - D. Dynamic type construction.
 - E. None of the above.
4. Which UML notation represents composition?
 - A. A filled diamond at the owning class.
 - B. A hollow triangle.
 - C. A dotted arrow.
 - D. A solid circle.
 - E. None of the above.
5. Which relationship best describes “a Car has an Engine”?
 - A. Composition.
 - B. Aggregation.
 - C. General association.
 - D. Inheritance.
 - E. None of the above.
6. Which relationship best describes “a Team has Players”?
 - A. Aggregation.
 - B. Composition.
 - C. Inheritance.
 - D. Association only.
 - E. None of the above.
7. Which relationship could be modeled as association in one design and composition in another?
 - A. “A Document contains Sections.”
 - B. “A Human has a Heart.”
 - C. “A Car has an Engine.”
 - D. “A Circle is a Shape.”
 - E. None of the above.
8. Which scenario BEST illustrates a misuse of inheritance?
 - A. Modeling “Car has Driver” using inheritance.

- B. Using association for unrelated objects.
 - C. Using composition for integral components.
 - D. Using aggregation for shared objects.
 - E. None of the above.
9. Which UML feature expresses multiplicity?
- A. Notation like $1..*$, $0..1$, $*$.
 - B. Dotted arrowheads.
 - C. Filled triangles.
 - D. Class stereotypes.
 - E. None of the above.
10. Which problem may occur if relationships are incorrectly modeled?
- A. Confusing lifetimes or ownership semantics.
 - B. Faster execution with reduced safety.
 - C. Compiler errors unrelated to design.
 - D. Automatic enforcement of aggregation.
 - E. None of the above.
11. Which relationship implies shared references but not ownership?
- A. Aggregation.
 - B. Composition.
 - C. Inheritance.
 - D. Realization.
 - E. None of the above.
12. Which option demonstrates composition in C++?
- A. A class storing another object as a direct member (not via pointer).
 - B. A class storing a pointer to an external resource.
 - C. Storing all members in `std::shared_ptr`. *Using a base pointer to refer to derived objects.*
 - D. None of the above.

Perfect, Terry — now that you've patched the minted issue with “ (excellent move), we can continue seamlessly.

Topic 16 — Generics & Templates

Total Questions: 12

You must answer at least 7.

You may skip up to 5 questions.

You may mark ONE question with a Y-token for double credit.

1. What is the primary purpose of templates in C++?
 - A. To enable type-agnostic programming with compile-time type safety.
 - B. To replace inheritance for all polymorphic behavior.
 - C. To force dynamic dispatch at runtime.
 - D. To eliminate header files entirely.
 - E. None of the above.
2. Which statement best differentiates templates from macros?
 - A. Templates are type-checked and instantiated by the compiler; macros are purely text substitution.
 - B. Macros enforce compile-time type safety.
 - C. Templates run only at runtime.
 - D. Templates cannot generate multiple functions.
 - E. Both A and C.
3. Which Python feature resembles generics?
 - A. Type hints using `list[int]`, `dict[str, float]`, and `TypeVar`.
 - B. Python templates declared with `template<T>`.
 - C. Python macros via `#define`.
 - D. Python requires explicit specialization for each generic type.
 - E. None of the above.
4. Template instantiation occurs:
 - A. When the compiler encounters a use of the template with specific types.
 - B. Only during linking.
 - C. Only if the programmer writes `instantiate<T>()`.
 - D. Before preprocessing begins.
 - E. None of the above.
5. Which advantage is commonly associated with C++ templates?
 - A. Zero-cost abstraction through compile-time polymorphism.
 - B. Guaranteed faster compile times.
 - C. Simplified debugging.
 - D. Automated memory management.
 - E. All of the above.
6. What is template specialization used for?
 - A. Providing custom behavior for a specific type argument.
 - B. Preventing template instantiation.
 - C. Forcing dynamic dispatch.
 - D. Removing generic type constraints.
 - E. None of the above.
7. In Python, what does `TypeVar` accomplish?
 - A. Declares that a function or class can operate over a range of possible types.
 - B. Enforces strict runtime type constraints.
 - C. Automatically optimizes generic calls.
 - D. Enables compile-time template expansion.
 - E. None of the above.
8. Why are template error messages often difficult for beginners?
 - A. Instantiation generates complex, nested types that obscure the original error.

- B. Templates have no type-checking.
 - C. Compilers hide all template-related diagnostics.
 - D. Templates execute at runtime.
 - E. All of the above.
9. Given `template<class T> T max(T a, T b)`, which constraint must hold?
- A. T must support comparison with < or similar.
 - B. T must be numeric.
 - C. T must be a pointer type.
 - D. T must be trivially copyable.
 - E. None of the above.
10. Python's generic type parameters are:
- A. Erased at runtime, serving mainly for static analysis.
 - B. Enforced strictly at runtime.
 - C. Stored in each object's metadata.
 - D. Used to specialize bytecode.
 - E. None of the above.
11. Templates differ from inheritance-based polymorphism because:
- A. Templates provide compile-time polymorphism, while inheritance provides runtime polymorphism.
 - B. Templates always use vtables.
 - C. Inheritance performs compile-time dispatch.
 - D. Templates cannot be used with classes.
 - E. None of the above.
12. Which scenario represents poor use of templates?
- A. Creating deeply nested, unreadable template chains for trivial tasks.
 - B. Building generic containers.
 - C. Specializing behavior for known edge cases.
 - D. Using templates in combination with composition.
 - E. None of the above.

Topic 17 — Collections & Iterators

Total Questions: 12

You must answer at least 7.

You may skip up to 5 questions.

You may mark ONE question with a Y-token for double credit.

1. Which statement best defines a collection?
 - A. A data structure that stores multiple elements and supports iteration.
 - B. Any object allocated on the heap.
 - C. Any class with two or more fields.
 - D. Any container requiring templates.
 - E. None of the above.
2. Why is the C++ range-based for loop preferred?
 - A. It reduces errors and works with any container providing `begin()` and `end()`.
 - B. It automatically parallelizes.
 - C. It prevents iterator invalidation.
 - D. It requires no copying.
 - E. None of the above.
3. How does Python's `for` loop relate to C++ iteration?
 - A. Both rely on iterator-like protocols behind the scenes.
 - B. Python requires explicit iterators for all loops.
 - C. Python loops only over lists.
 - D. Python forcing copy semantics.
 - E. None of the above.
4. What is the main benefit of using `auto` in loops?
 - A. The compiler deduces iterator or element type automatically.
 - B. It disables copying.
 - C. It forces reference semantics.
 - D. It ensures faster runtime.
 - E. None of the above.
5. Which scenario commonly invalidates iterators?
 - A. Inserting/erasing elements in vectors that may reallocate.
 - B. Iterating through `const` containers.
 - C. Using `auto`.
 - D. Passing iterators by reference.
 - E. None of the above.
6. Which C++ collection is best for random access?
 - A. `std::vector`
 - B. `std::list`
 - C. `std::deque`
 - D. `std::set`
 - E. All of the above.
7. Which statement about Python lists vs C++ vectors is correct?
 - A. Python lists support mixed types; C++ vectors do not.
 - B. Both are stored contiguously.
 - C. Both require fixed capacity.
 - D. Lists always allocate on the stack.
 - E. None of the above.
8. When should explicit iterators be used?
 - A. When performing fine-grained traversal or conditional erasure.

- B. Always.
 - C. Only for raw arrays.
 - D. Never; iterators are outdated.
 - E. None of the above.
9. Why must collections implement `begin()` and `end()`?
- A. Range-based for loops expand to use them.
 - B. For template instantiation.
 - C. For operator overloading.
 - D. For memory alignment.
 - E. None of the above.
10. What does `const` iteration enforce?
- A. Immutable access to container elements.
 - B. Faster iteration.
 - C. No copying.
 - D. Reverse traversal only.
 - E. None of the above.
11. Which demonstrates misuse of `auto`?
- A. Accidentally deducing a reference type and causing lifetime bugs.
 - B. Using `auto` for long iterator types.
 - C. Using `auto` in loops.
 - D. Using `auto` for lambdas.
 - E. None of the above.
12. Which is a known pitfall of range-based loops?
- A. Iterating by value unintentionally, causing expensive copies.
 - B. Losing access to iterators.
 - C. Preventing exception safety.
 - D. Disabling polymorphism.
 - E. None of the above.

Topic 18 — Memory Management

Total Questions: 14

You must answer at least 8.

You may skip up to 6 questions.

You may mark ONE question with a Y-token for double credit.

1. Which statement correctly describes stack vs heap allocation?
 - A. Stack allocation is automatic and deterministic; heap allocation requires explicit management or smart pointers.
 - B. Heap is always faster.
 - C. Stack memory persists until program termination.
 - D. Heap allocation guarantees no fragmentation.
 - E. None of the above.
2. Why are raw pointers dangerous?
 - A. They do not express ownership semantics and require manual deletion.
 - B. They cannot store memory addresses.
 - C. They ensure automatic deletion.
 - D. They prevent copying.
 - E. None of the above.
3. Which statement best describes `std::unique_ptr`?
 - A. Exclusive ownership with automatic deletion.
 - B. Shared ownership.
 - C. Garbage-collected behavior.
 - D. Requires manual deletion.
 - E. None of the above.
4. Why can `std::shared_ptr` leak memory?
 - A. Reference cycles prevent counts from reaching zero.
 - B. It deletes too early.
 - C. It forces move semantics.
 - D. It cannot store pointers to heap memory.
 - E. None of the above.
5. Which statement describes Python memory management?
 - A. Reference counting with periodic garbage collection for cycles.
 - B. Deterministic destruction.
 - C. Manual `delete`.
 - D. Static memory only.
 - E. None of the above.
6. Which scenario leads to a dangling pointer?
 - A. Deleting an object but continuing to use its pointer.
 - B. Returning objects by value.
 - C. Using smart pointers.
 - D. Using `auto`.
 - E. None of the above.
7. What commonly causes memory leaks in C++?
 - A. Forgetting to `delete` memory allocated with `new`.
 - B. Returning references.
 - C. Using RAII.
 - D. Using smart pointers correctly.
 - E. None of the above.
8. How do move semantics improve performance?

- A. By transferring ownership instead of copying resources.
 - B. By deleting source objects automatically.
 - C. By forcing stack allocation.
 - D. By preventing virtual dispatch.
 - E. None of the above.
9. The Rule of Three/Five applies when:
- A. A class manages resources that require manual cleanup.
 - B. A class uses only pure virtual functions.
 - C. No heap allocation occurs.
 - D. Python interoperability is required.
 - E. None of the above.
10. A limitation of Python's garbage collector is:
- A. Cycles may persist until the cycle detector runs.
 - B. It cannot collect unreachable objects.
 - C. It guarantees deterministic cleanup.
 - D. It exposes manual alloc/free.
 - E. None of the above.
11. Why does RAII improve memory safety?
- A. It ties resource lifetime to object lifetime, ensuring cleanup even during exceptions.
 - B. It disables virtual destructors.
 - C. It eliminates all pointer usage.
 - D. It forces heap allocation for all objects.
 - E. None of the above.
12. Which scenario illustrates a shallow copy issue?
- A. Double deletion due to two objects owning the same pointer.
 - B. Unique ownership using `unique_ptr`.*Copying integers.*
 - C. Storing objects in a vector.
 - D. None of the above.
13. What does placement `new` do?
- A. Constructs an object at a specific pre-allocated memory address.
 - B. Allocates memory automatically.
 - C. Skips constructors.
 - D. Automatically frees previous objects.
 - E. None of the above.
14. When is `std::shared_ptr` or `choice`?
- A. When exclusive ownership is clear and shared ownership adds overhead or confusion.
 - B. When multiple owners legitimately exist.
 - C. When thread-safe reference counting matters.
 - D. When objects require automatic deletion.
 - E. None of the above.

Topic 19 — UML Diagrams & Modeling

Total Questions: 12

You must answer at least 7.

You may skip up to 5 questions.

You may mark ONE question with a Y-token for double credit.

1. Which UML diagram is MOST appropriate for describing system behavior over time?
 - A. Sequence diagram.
 - B. Class diagram.
 - C. Use-case diagram.
 - D. Deployment diagram.
 - E. None of the above.
2. In UML class diagrams, a filled diamond represents:
 - A. Composition (strong ownership).
 - B. Aggregation (shared ownership).
 - C. Inheritance.
 - D. Dependency.
 - E. None of the above.
3. Which UML modeling element represents an interface?
 - A. A class with the stereotype <<interface>>.
 - B. A class with dashed borders.
 - C. A hollow diamond.
 - D. A deployment node.
 - E. None of the above.
4. What is the purpose of multiplicity such as $1..*$?
 - A. Specifies how many instances participate in a relationship.
 - B. Specifies method overloading rules.
 - C. Ensures composition semantics.
 - D. Indicates internal visibility.
 - E. None of the above.
5. Which UML diagram type best models high-level user interactions?
 - A. Use-case diagram.
 - B. Object diagram.
 - C. Sequence diagram.
 - D. Package diagram.
 - E. None of the above.
6. Which relationship is shown with a hollow triangle arrow?
 - A. Inheritance (generalization).
 - B. Realization.
 - C. Composition.
 - D. Aggregation.
 - E. None of the above.
7. What does a dashed arrow with an open arrowhead typically represent?
 - A. Dependency.
 - B. Inheritance.
 - C. Composition.
 - D. Aggregation.
 - E. None of the above.
8. Which choice best describes why UML is useful in OOP?
 - A. It provides a common visual language to represent structure and behavior.

- B. It verifies code correctness.
 - C. It eliminates the need for documentation.
 - D. It enforces polymorphism.
 - E. None of the above.
9. In a sequence diagram, what does a lifeline represent?
- A. The time span during which an object exists and can participate in interactions.
 - B. A class definition.
 - C. A composition relationship.
 - D. An interface boundary.
 - E. None of the above.
10. Which UML diagram reveals object creation and destruction explicitly?
- A. Sequence diagram.
 - B. Class diagram.
 - C. Use-case diagram.
 - D. Deployment diagram.
 - E. None of the above.
11. Which UML concept distinguishes “type” from “instance”?
- A. Class diagram vs. object diagram.
 - B. Package diagram vs. sequence diagram.
 - C. Deployment diagram vs. profile.
 - D. State machine vs. activity diagram.
 - E. None of the above.
12. What is a common misuse of UML?
- A. Treating diagrams as exact code rather than communication tools.
 - B. Using arrows to represent flow.
 - C. Using multiple diagram types.
 - D. Omitting stereotypes.
 - E. None of the above.

Topic 20 — Refactoring & Code Smells

Total Questions: 14

You must answer at least 8.

You may skip up to 6 questions.

You may mark ONE question with a Y-token for double credit.

1. Which option best defines a “code smell”?
 - A. A surface-level symptom indicating deeper design problems.
 - B. A syntax error.
 - C. A compiler warning.
 - D. Poor indentation.
 - E. None of the above.
2. Long methods, large classes, and feature envy are examples of:
 - A. Common code smells.
 - B. Poor testing practices.
 - C. Issues solved automatically by inheritance.
 - D. Violations of Python conventions only.
 - E. None of the above.
3. Refactoring focuses primarily on:
 - A. Improving internal design without changing external behavior.
 - B. Optimizing runtime performance.
 - C. Altering class interfaces.
 - D. Removing polymorphism.
 - E. None of the above.
4. Which smell is characterized by too many responsibilities in one class?
 - A. God Object.
 - B. Dead Code.
 - C. Message Chain.
 - D. Switch Statements.
 - E. None of the above.
5. Which refactoring best addresses duplicated code?
 - A. Extract Method.
 - B. Inline Temp.
 - C. Introduce Null Object.
 - D. Move Method.
 - E. None of the above.
6. A “shotgun surgery” smell occurs when:
 - A. One small change forces modifications across many unrelated classes.
 - B. A class contains unused imports.
 - C. Code is tightly cohesive.
 - D. A method is too short.
 - E. None of the above.
7. Python’s duck typing can hide which code smell?
 - A. Inconsistent interfaces.
 - B. Long method.
 - C. Dead code.
 - D. Message chains.
 - E. None of the above.
8. Which refactoring reduces the complexity of nested conditionals?
 - A. Replace Conditional with Polymorphism.

- B. Inline Method.
 - C. Remove Dead Code.
 - D. Extract Variable.
 - E. None of the above.
9. Which code smell often indicates poor encapsulation?
- A. Feature Envy.
 - B. Primitive Obsession.
 - C. God Class.
 - D. Data Class.
 - E. None of the above.
10. A sign of “data clumps” is:
- A. The same set of variables repeatedly passed together.
 - B. Long parameter lists with unrelated values.
 - C. A class with only getters.
 - D. Removing abstraction.
 - E. None of the above.
11. Which refactoring is used to separate responsibilities across new classes?
- A. Extract Class.
 - B. Push Down Method.
 - C. Inline Class.
 - D. Rename Method.
 - E. None of the above.
12. What is a key risk of premature optimization?
- A. It can distort design clarity and introduce unnecessary complexity.
 - B. It always improves maintainability.
 - C. It removes polymorphic overhead safely.
 - D. It simplifies inheritance.
 - E. None of the above.
13. Which situation BEST represents “dead code”?
- A. Code that is never executed or referenced.
 - B. Code that runs too slowly.
 - C. Code that uses recursion.
 - D. Code that changes state.
 - E. None of the above.
14. Which statement accurately reflects refactoring strategy?
- A. Refactor in small, verifiable steps with tests ensuring correctness.
 - B. Refactor only after full rewrites.
 - C. Refactoring should be avoided near deadlines.
 - D. All refactorings improve performance.
 - E. None of the above.

Topic 21 — SOLID Principles

Total Questions: 12

You must answer at least 7.

You may skip up to 5 questions.

You may mark ONE question with a Y-token for double credit.

1. The Single Responsibility Principle (SRP) states that:
 - A. A class should have one and only one reason to change.
 - B. Every method must be under 10 lines.
 - C. Classes should restrict inheritance.
 - D. All methods must be static.
 - E. None of the above.
2. The Open-Closed Principle (OCP) promotes:
 - A. Software entities should be open for extension but closed for modification.
 - B. Classes should never be extended.
 - C. Only inheritance-based polymorphism.
 - D. Forcing static dispatch.
 - E. None of the above.
3. Which scenario violates Liskov Substitution Principle (LSP)?
 - A. A derived class overrides behavior in a way that breaks expected base-class guarantees.
 - B. A base class defines a virtual method.
 - C. A subclass adds new optional parameters.
 - D. Using templates instead of inheritance.
 - E. None of the above.
4. The Interface Segregation Principle (ISP) argues that:
 - A. Clients should not be forced to depend on methods they do not use.
 - B. Interfaces must contain at least 10 functions.
 - C. Multiple inheritance must be avoided universally.
 - D. All abstract classes are interfaces.
 - E. None of the above.
5. Which example best represents Dependency Inversion Principle (DIP)?
 - A. High-level modules depend on abstractions rather than concrete implementations.
 - B. All modules depend directly on each other.
 - C. High-level modules create their own dependencies.
 - D. Using global variables.
 - E. None of the above.
6. A common Python violation of SRP is:
 - A. A class acting as data storage, validator, serializer, and logger simultaneously.
 - B. Using decorators.
 - C. Using `init` for attribute assignment. Returning dictionaries.
 - D. None of the above.
7. Why does LSP matter in OOP?
 - A. Violating LSP breaks polymorphism and substitutability.
 - B. It ensures faster performance.
 - C. It decreases testing needs.
 - D. It prevents abstraction.
 - E. None of the above.
8. Which of the following shows strong adherence to OCP?
 - A. Introducing new subclasses instead of modifying stable code.
 - B. Editing a widely-used base class for each new feature.

- C. Copy/pasting existing classes.
 - D. Using macros.
 - E. None of the above.
9. Which of the SOLID principles directly combats “fat” interfaces?
- A. ISP.
 - B. SRP.
 - C. LSP.
 - D. DIP.
 - E. None of the above.
10. Which design outcome aligns with DIP?
- A. Injecting dependencies (e.g., passing an abstract interface rather than constructing inside).
 - B. Hardcoding object creation.
 - C. Using static utility classes.
 - D. Returning primitive types.
 - E. None of the above.
11. Which principle emphasizes separating concerns into small, cohesive parts?
- A. SRP.
 - B. OCP.
 - C. LSP.
 - D. ISP.
 - E. None of the above.
12. Which pattern strongly supports DIP?
- A. Dependency Injection pattern.
 - B. Singleton.
 - C. Factory Method.
 - D. Observer.
 - E. None of the above.

Topic 22 — Testing & Test-Driven Development (TDD)

Total Questions: 12

You must answer at least 7.

You may skip up to 5 questions.

You may mark ONE question with a Y-token for double credit.

1. Which statement best describes unit testing?
 - A. Testing individual functions or classes in isolation.
 - B. Testing the entire system as a whole.
 - C. Testing at the network boundary.
 - D. Stress testing I/O devices.
 - E. None of the above.
2. Which sequence correctly represents the TDD cycle?
 - A. Red → Green → Refactor.
 - B. Green → Red → Refactor.
 - C. Implement → Test → Debug.
 - D. Refactor → Implement → Test.
 - E. None of the above.
3. Why does TDD typically improve design quality?
 - A. Tests force smaller, more cohesive units with clear responsibilities.
 - B. TDD eliminates the need for refactoring.
 - C. TDD guarantees faster runtime performance.
 - D. TDD requires inheritance.
 - E. None of the above.
4. What is a test fixture?
 - A. A reusable setup that prepares objects or state for tests.
 - B. A test that depends on external hardware.
 - C. A single assert statement.
 - D. A test that benchmarks performance.
 - E. None of the above.
5. Which principle is violated when tests rely on external databases or network calls?
 - A. Tests should be deterministic and isolated.
 - B. Tests should always be written last.
 - C. Tests must depend on production data.
 - D. Tests must operate without mocking.
 - E. None of the above.
6. In Python's `unittest`, what does `setUp()` provide?
 - A. A method executed before each test to prepare state.
 - B. A method executed after all tests.
 - C. A method that runs only when tests fail.
 - D. A teardown function for cleaning up mocks.
 - E. None of the above.
7. What makes tests brittle?
 - A. Depending on implementation details instead of public behavior.
 - B. Testing only behavior visible to clients.
 - C. Mocking external dependencies.
 - D. Using descriptive test names.
 - E. None of the above.
8. Which type of test is MOST likely to reveal integration problems early?
 - A. Integration tests combining multiple components.

- B. Unit tests of isolated functions.
 - C. Static code analysis.
 - D. Comments describing expected behavior.
 - E. None of the above.
9. Which statement reflects a benefit of mocks or stubs?
- A. They replace slow or external components, making tests fast and deterministic.
 - B. They eliminate the need for assertions.
 - C. They prevent refactoring.
 - D. They enforce inheritance hierarchies.
 - E. None of the above.
10. A failing test that should never fail indicates:
- A. A regression or unintended behavior change.
 - B. The test should be deleted.
 - C. Tests should be disabled temporarily.
 - D. Tests should avoid using expected values.
 - E. None of the above.
11. Which scenario undermines TDD?
- A. Writing tests only after implementation stabilizes.
 - B. Writing minimal failing tests.
 - C. Refactoring after tests pass.
 - D. Using a mocking library.
 - E. None of the above.
12. Which is a reasonable goal of test coverage?
- A. High coverage while avoiding meaningless or redundant tests.
 - B. 100
 - C. Only testing error paths.
 - D. No unit tests for pure functions.
 - E. None of the above.

Topic 23 — Object-Oriented Design Principles

Total Questions: 14

You must answer at least 8.

You may skip up to 6 questions.

You may mark ONE question with a Y-token for double credit.

1. Which principle emphasizes designing software to be understandable and maintainable first?
 - A. Readability and clarity should trump premature optimization.
 - B. Low-level memory management is always the priority.
 - C. Deep inheritance chains are preferred.
 - D. Performance determines design choices exclusively.
 - E. None of the above.
2. Which principle encourages using composition over inheritance?
 - A. Favor objects that delegate behavior rather than inheriting it.
 - B. Avoid creating helper classes.
 - C. Always inherit from at least two base classes.
 - D. Never use interfaces.
 - E. None of the above.
3. Which statement about cohesion is correct?
 - A. High cohesion means a class's responsibilities are tightly related.
 - B. High cohesion means a class has many unrelated methods.
 - C. Low cohesion improves testing.
 - D. Cohesion applies only to modules, not classes.
 - E. None of the above.
4. Which design issue often results from deep inheritance hierarchies?
 - A. Fragility and difficulty extending behavior safely.
 - B. Tighter encapsulation.
 - C. Reduced coupling.
 - D. Fewer dependencies.
 - E. None of the above.
5. What does coupling refer to?
 - A. How strongly components depend on each other.
 - B. How many fields a class has.
 - C. How often constructors run.
 - D. Whether an object uses polymorphism.
 - E. None of the above.
6. Which design approach intentionally hides implementation details?
 - A. Encapsulation.
 - B. Abstraction.
 - C. Inheritance.
 - D. Polymorphism.
 - E. None of the above.
7. Which is MOST likely to cause tight coupling?
 - A. A class instantiating its own dependencies instead of receiving them.
 - B. Using interfaces.
 - C. Passing behavior via function objects.
 - D. Clear separation of concerns.
 - E. None of the above.
8. Which principle suggests that classes should expose small, minimal public interfaces?
 - A. Encapsulation + Interface Minimalism.

- B. Always overloading operators.
 - C. Avoiding private members.
 - D. Using global variables.
 - E. None of the above.
9. Which practice improves maintainability?
- A. Documenting *why* code exists, not just how it works.
 - B. Embedding all documentation in comments only.
 - C. Removing tests.
 - D. Writing methods over 200 lines.
 - E. None of the above.
10. Which principle is violated when unrelated responsibilities get mixed into one class?
- A. SRP (Single Responsibility Principle).
 - B. DIP.
 - C. LSP.
 - D. ISP.
 - E. None of the above.
11. Which approach enhances code extensibility?
- A. Program to an interface, not an implementation.
 - B. Hardcoding types inside methods.
 - C. Rewriting entire modules when adding features.
 - D. Using global state.
 - E. None of the above.
12. Which practice avoids accidental behavior changes during refactoring?
- A. Writing tests before modifying implementation.
 - B. Only refactor when code breaks.
 - C. Disable regression tests.
 - D. Avoid abstraction.
 - E. None of the above.
13. What is a hallmark of a maintainable object model?
- A. Independent modules that communicate through stable, minimal interfaces.
 - B. Deep inheritance hierarchies.
 - C. Heavy static state.
 - D. Classes with ambiguous responsibilities.
 - E. None of the above.
14. Which scenario best reflects good object-oriented decomposition?
- A. Splitting a large, multi-purpose class into focused collaborating objects.
 - B. Collapsing many classes into a single “manager.”
 - C. Using inheritance solely to reduce typing.
 - D. Duplicating logic across unrelated modules.
 - E. None of the above.