🔹 Junior-Level C++ Interview Questions — Professional Answers

---

1. **What are the key features of object-oriented programming in C++?**

C++ supports core OOP features:

Encapsulation: Binding data and functions into a class.

Abstraction: Hiding implementation details using access specifiers and interfaces.

Inheritance: One class acquires properties of another, enabling reusability.

Polymorphism: Same interface, different behavior (compile-time via overloading; runtime via virtual functions).

---

2. **Explain the differences between public, private, and protected access specifiers in C++.**

Public: Members are accessible from anywhere.

Private: Members are accessible only within the class.

Protected: Members are accessible within the class and by derived classes.

---

3. **Distinguish between function overloading and overriding in C++.**

Overloading: Same function name with different parameters within the same scope.

Overriding: Derived class redefines a base class's virtual method to provide specific behavior.

---

4. **Compare and contrast abstract classes and interfaces in C++.**

C++ doesn't have "interfaces" like Java, but:

Abstract class: Contains at least one pure virtual function.

Used as a base for implementing polymorphic behavior.

Interfaces in C++ can be simulated using classes with only pure virtual methods and no data members.

---

5. **Can an interface inherit from another interface in C++?**

Yes, an abstract class (used as interface) can inherit from another abstract class using virtual inheritance.

---

6. **Define the static keyword in C++ and its significance.**

For variables: Shared across all instances.

For functions: Belong to the class, not the object; can be called without creating an object.

For local variables: Retains value between function calls.

---

7. **Is it possible to override a static method in C++?**

No. Static methods are not tied to an object instance and are resolved at compile-time, hence not subject to polymorphism.

---

8. **Explain the concepts of polymorphism and inheritance in C++.**

Inheritance: Deriving a new class from an existing one. Promotes code reuse.

Polymorphism: Allows function/methods to behave differently based on context.

Compile-time: Function overloading, operator overloading

Runtime: Virtual functions and dynamic binding

---

9. **Can constructors be inherited in C++?**

Constructors are not inherited, but C++11 allows using using Base::Base; in derived classes to inherit base constructors.

---

10. **Discuss pass-by-reference and pass-by-value for objects in C++.**

Pass-by-value: Copies the object; changes in the function do not affect the original.

Pass-by-reference: Passes a reference; changes reflect in the original object.

---

11. **Compare == and .equals for string comparison in C++.**

In C++, you compare strings using:

== for std::string (compares content).

There is no .equals() method like Java; content comparison is directly supported via ==.

---

12. **Explain the purposes of the hashCode() and equals() functions**.

C++ doesn't have hashCode()/equals() like Java, but:

Std::hash is used in unordered containers.

Custom comparison or hash functions can be defined for objects when used in unordered_map.

---

13. **What does the Serializable interface do? How is it related to Parcelable in Android?**

This is Java-specific. In C++, serialization is done manually or with libraries like Boost.Serialization, cereal, or protobuf.

C++ doesn't have a standard Serializable or Parcelable interface like Java or Android.

---

14. **Differentiate between Array and ArrayList in C++. When would you use each?**

C++ uses array (fixed size) and std::vector (dynamic array).

Use std::array when size is known and fixed.

Use std::vector when size can change during runtime.

---

15. **Explain the distinction between Integer and int in C++.**

This is Java terminology. In C++:

Int is a primitive type.

C++ doesn't have boxed primitive types like Integer, but you can wrap primitives in classes if needed.

---

16. **Define ThreadPool and discuss its advantages over using simple threads.**

ThreadPool is a pool of worker threads reused to execute tasks, reducing overhead.

Advantages:

Faster response by reusing threads.

Limits thread creation overhead.

Better resource control.

In C++, std::thread and libraries like Boost::asio, ThreadPool, or custom implementations are used.

---

17. **Differentiate between local, instance, and class variables in C++.**

Local variables: Declared inside functions or blocks; limited scope.

Instance variables: Non-static class members; unique to each object.

Class variables: Static members; shared among all instances.

🔹 Mid-Level C++ Interview Questions — Professional Answers

---

1. **What is reflection in C++?**

C++ does not natively support reflection like Java or C#. However, reflection-like behavior can be simulated using:

RTTI (Run-Time Type Information) with typeid and dynamic_cast

External libraries such as RTTR, Boost.TypeIndex, or meta-programming techniques in modern C++

---

2. **Define dependency injection and name a few libraries. Have you used any?**

Dependency Injection (DI) is a design pattern where a class's dependencies are provided from the outside rather than created internally.

In C++, it's typically done via:

Constructor injection

Setter injectionn

Service locators

Popular libraries for DI:

Boost.DI

Fruit

Autowiring

DI promotes loose coupling and testability.

---

3. **Explain strong, soft, and weak references in C++.**

In C++ (specifically C++11 onward), memory management uses smart pointers:

Std::shared_ptr (strong reference): Maintains reference count, object destroyed when count hits zero.

Std::weak_ptr (weak reference): Observes a shared_ptr without affecting its lifetime.

Soft references are not natively supported; they're more common in Java.

---

4. **Interpret the meaning of the synchronized keyword.**

Synchronized is a Java keyword. In C++, thread safety is handled with:

Std::mutex, std::lock_guard, and std::unique_lock Used to protect shared data in multithreading environments.

---

5. **Can memory leaks occur in C++?**

Yes. C++ allows manual memory management (new, delete), which can lead to leaks if:

Dynamically allocated memory is not freed.

References are lost before deletion.

Using RAII (Resource Acquisition Is Initialization) and smart pointers (unique_ptr, shared_ptr) prevents most leaks.

---

6. **Is it necessary to set references to null in C++?**

C++ references (int&, Object&) cannot be null. You cannot reset them to null.

However, pointers can and often should be set to nullptr after deletion to avoid dangling pointer issues.

---

7. **Why is a std::string considered immutable?**

C++'s std::string is not immutable like Java's String.

It is mutable: You can modify its contents.

However, C++ implementations use copy-on-write or small string optimizations to reduce overhead.

---

8. **Discuss transient and volatile modifiers in C++.**

Volatile in C++ tells the compiler that a variable may change outside program control (e.g., hardware access or multithreading).

Transient is not available in C++; it's a Java concept used during serialization to skip fields.

---

9. **What is the purpose of the finalize() method?**

Finalize() is a Java concept, not available in C++.

In C++, destructors (~ClassName()) serve the same purpose — to release resources when an object is destroyed.

---

10. How does the try{} finally{} block work in C++?

C++ uses try/catch blocks for exception handling, but does not have finally.

Instead, it uses RAII: Objects automatically clean up in destructors when going out of scope, ensuring resource deallocation even during exceptions.

---

11. Explain the difference between object instantiation and initialization.

Instantiation: Creating an object in memory.

Initialization: Assigning a value/state to the newly created object (via constructor, etc.).

In C++:

MyClass obj; // Instantiation + default initialization

MyClass obj(10); // Instantiation + parameterized initialization

---

12. Under what conditions is a static block executed in C++?

C++ doesn’t have a “static block” like Java, but:

Static local variables are initialized the first time the function is called.

Static global variables are initialized once at program start, before main().

---

13. Why are generics used in C++?

C++ uses templates to achieve generic programming:

Functions or classes that work with any data type.

Enables type safety, code reusability, and performance (since resolved at compile-time).

Example:

```
Template <typename T>
T add(T a, T b) {
  Return a + b;
}
```

---

14. Mention some design patterns you are familiar with. Which do you typically use?

Common patterns in C++:

Singleton

Factory

Observer

Strategy

Adapter

RAII (unique to C++)

Most used in real-world C++: Factory for object creation, Singleton for shared state, and RAII for resource safety.

---

15. Name some types of testing methodologies in C++.

Unit testing: With frameworks like Google Test or Catch2

Integration testing: Verifies modules/components together

System testing: Full application validation

Mock testing: With libraries like Trompeloeil, FakeIt

Static code analysis: Tools like cppcheck, Clang-Tidy

🔹 Senior-Level C++ Interview Questions — Professional Answers

---

1. **Explain how std::stoi (string to integer) works in C++.**

Std::stoi converts a string to an int by parsing the string and interpreting its contents as a base-10 integer.

It throws:

Std::invalid_argument if the input isn't a number

Std::out_of_range if the result doesn't fit in an int

Example:

Int num = std::stoi("123"); // returns 123

---

2. **What is the "double-checked locking" problem, and how can it be solved in C++?**

It's a concurrency issue when initializing a singleton:

```
If (!instance) {
   Lock(mutex);
   If (!instance) {
      Instance = new Singleton();
   }
}
```

Problem: CPU instruction reordering can lead to using a partially constructed object.

Solution:

Use std::atomic

Prefer std::call_once with std::once_flag in C++11 and beyond for thread-safe initialization.

---

3. **Differentiate between StringBuffer and StringBuilder in C++.**

These are Java classes. In C++:

Use std::ostringstream for efficient string concatenation.

Std::stringstream is not thread-safe (like StringBuilder). For thread-safe behavior, use mutexes or thread-local storage.

---

4. **How is StringBuilder implemented to avoid the immutable string allocation problem?**

StringBuilder (Java) uses a mutable internal buffer to prevent repeated allocation.

In C++, std::ostringstream uses a similar approach: it manages a resizable internal buffer, allowing efficient concatenation without creating new strings each time.

---

5. **Explain the purpose of the Class.forName method in C++.**

Class.forName is a Java reflection feature.

C++ has no direct equivalent, but similar behavior is possible using:

Factory design pattern

Registration maps with function pointers or lambdas

External tools like RTTR for reflection-like functionality

---

6. **Define Autoboxing and Unboxing in C++.**

These are Java concepts.

In C++, there's no autoboxing — you explicitly wrap primitives in classes.

However, you can simulate it via:

```
Class Integer {
   Int value;
Public:
   Integer(int v) : value(v) {}
   Operator int() const { return value; }
};
```

---

7. **What's the difference between Enumeration and Iterator in C++?**

C++ uses:

Iterators for STL containers (vector, list, map, etc.)

Enumerators aren't built-in for containers but enums exist for constants.

Iterators can be:

Input/Output

Forward/Bidirectional/Random access

They support algorithms like std::for_each, std::copy, etc.

---

8. **Explain the difference between fail-fast and fail-safe in C++.**

Fail-fast: Iterators like in std::vector become invalid if the container is modified during iteration — may cause crashes.

Fail-safe: Not natively supported. Some third-party libraries offer this by working on copies or with concurrent structures.

---

9. **What is PermGen in C++?**

PermGen is a Java-specific memory area for class metadata (deprecated in Java 8).

C++ doesn't have a PermGen.

Class metadata and type info are part of the binary, not managed by a runtime like in Java.

---

10. **Describe a Java priority queue.**

In C++, the equivalent is:

Std::priority_queue<int> pq; // Max-heap by default

Implemented using a binary heap (via std::make_heap, std::push_heap, etc.)

You can customize it for min-heap using a comparator.

---

11. **How is performance influenced by using the same number in different types: int, double, and float?**

Int: Fastest for integer operations.

Float: Lower precision, faster than double on some systems.

Double: Higher precision, slightly slower.

Trade-offs depend on CPU architecture, cache alignment, and compiler optimizations. In performance-critical code, use the type that best matches your data.

---

12. Explain the concept of the Java Heap.

In C++, heap refers to dynamically allocated memory using new, managed manually or via smart pointers.

Unlike Java:

No garbage collector.

Programmer is responsible for memory cleanup (RAII is preferred).

---

13. What is a daemon thread?

In Java, a daemon thread runs in the background.

C++ doesn't have a concept of daemon threads per se, but you can:

Detach a thread using .detach()

Mark a thread to run independently and not block main thread exit.

Std::thread([]{ /* work */ }).detach();

---

14. Can a dead thread be restarted in C++?

No.

Once a std::thread has finished execution or joined, it cannot be restarted.

You must create a new thread instance to run the same logic again.