

Unique Pointers and Polymorphism Solutions

Polymorphism with unique_ptr

- How can unique_ptr be used with polymorphism?
 - Instead of calling new() to get a pointer, call make_unique() instead
- What are the advantages of using unique_ptr?
 - The allocated memory will be managed by unique_ptr
 - A unique_ptr cannot be aliased
 - A unique_ptr cannot be accidentally reseated, overwritten or invalidated
 - A unique_ptr will automatically release the memory if an exception is thrown or when the object goes out of scope
 - A unique_ptr cannot be accidentally be used after releasing the memory

Polymorphic Container

- Using `unique_ptr`, write a program which creates a container of polymorphic objects and calls a member function on each object
- What are the advantages of this approach over using `new()`?
 - The memory is managed automatically by `unique_ptr`
 - The programmer does not need to consider when (or if) to call `delete()`

The Factory Pattern

- Briefly describe the factory pattern
 - In the factory pattern, a function creates a child class object and returns it through a pointer to base
- Briefly explain why the factory pattern is useful when working with class hierarchies
 - All the code to create objects in the hierarchy is in a single place
 - Flexibility - can use arguments to choose which child class type to create
 - Easy to extend if new child classes are added

Factory Pattern with unique_ptr

- Write a function that implements the factory pattern for the Shape hierarchy
- Add an argument to the function, so that the class of object created depends on the value of the argument
- Write a program to test your function