

CSE687 Midterm #1

Name: _____ **Instructor's Solution**_____

This is a closed book examination. Please place all your books on the floor beside you. You may keep one page of notes on your desktop in addition to this exam package. All examinations will be collected promptly at the end of the class period. Please be prepared to quickly hand in your examination at that time.

If you have any questions, please do not leave your seat. Raise your hand and I will come to your desk to discuss your question. I will answer all questions about the meaning of the wording of any question. I may choose not to answer other questions.

You will find it helpful to review all questions before beginning. All questions are given equal weight for grading, but not all questions have the same difficulty. Therefore, it is very much to your advantage to answer first those questions you believe to be easiest.

1. What does the term "hiding" mean in the context of polymorphic hierarchies? What implementation mistakes cause hiding?

Answer:

Hiding makes an inherited base class member function invisible at compile time in a derived class. This happens when either:

- a. A base class function is overloaded in a class derived from the base. C++ does not support overloading across class scopes. This is an error in the derived class.
- b. A virtual function is overloaded in a base class. When one of the overloads is redefined in a class derived from the base all of the other overloads are inherited but hidden. This is an error in the base class.

2. How would you ensure that a class implements a "create" function. Write all the code for an object factory that uses instances of these classes as prototypes for creating new objects of the same type.

Answer:

Provide an interface that declares the create function as pure virtual that classes for created instances must implement.

```
struct ICreatable
{
    virtual ICreatable* create()=0;
    virtual ~ICreatable() {}
};

class Factory
{
public:
    void registerPrototype(const std::string& key, ICreatable* prototype);
    ICreatable* createInstance(const std::string& key);
    ~Factory();
private:
    std::unordered_map<std::string, ICreatable*> map;
};

void Factory::registerPrototype(const std::string& key, ICreatable* ptr)
{
    map[key] = ptr;
}

ICreatable* Factory::createInstance(const std::string& key)
{
    return map[key]->create();
}

Factory::~~Factory()
{
    for(auto keyValuePair : map)
        delete keyValuePair.second;
}
```

3. Given the following code:

```
class X {  
    public:  
    ...  
    private:  
        std::ostream& out;  
};
```

Write a copy constructor for the class X, assuming the out is its only data member. Are there any issues with the behavior of your constructor?

Answer:

```
X::X(const X& x) : out(x.out) { }
```

Since out is a reference member, it must be initialized since it can't be set in the constructor body. Both instances must share the same ostream, which may or may not be desirable behavior for instances of this class.

Note that this is just what the compiler generated copy constructor does, so if you weren't answering an exam question you would write nothing.

4. Does the Dependency Inversion Principle apply to any parts of your implementation of Project #1¹? If so, discuss why it applies. If not, why not?

Answer:

Yes. Parser applies DIP, as Rules and Actions provide interfaces used by Parser and Rules, respectively, and configParser acts as an object factory. Parser has no binding to Rules implementation and Rules have no binding to Actions implementation. Note that Parser draws it's tokens from ITokenCollection and the Token and Semi instances are also created by configParser, so that too is an application of DIP.

There are other opportunities for DIP, e.g., the FileSystem could have provided an interface and object factory (but didn't).

The dependencies between Actions and Repository don't seem to fit with DIP since there doesn't seem to be a useful Repository interface language that would work across many different analysis applications.

¹ The Project #1 statement is attached at the end of this exam package.

5. Given the class declaration:

```
template<typename U, typename V>
class X { ... };
```

Show how you declare a specialization for `U = std::string`. Next provide a specialization declaration for `U = std::string` and `V = double`;

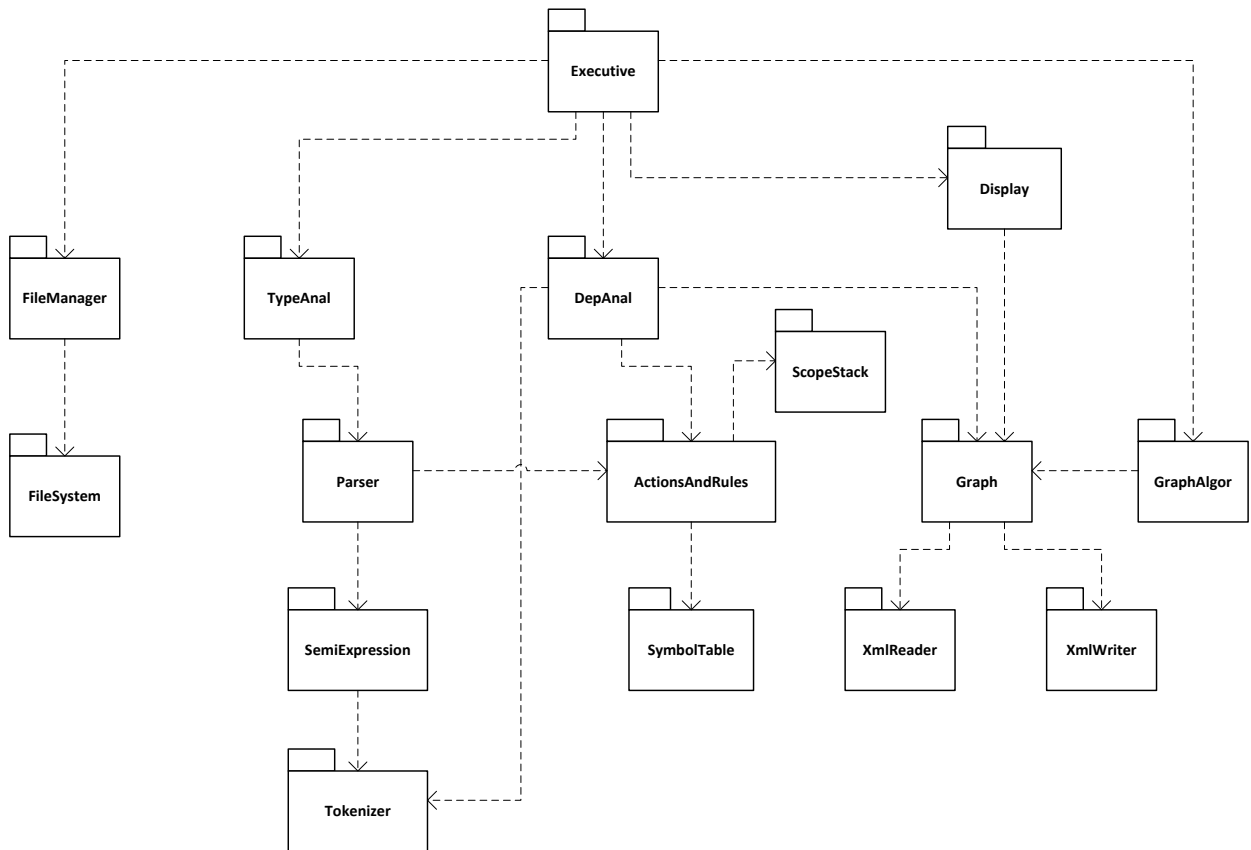
Answer:

```
template<typename V>
class X<std::string, V> { ... };

template<>
class X<std::string, double> { ... };
```

6. Draw a package diagram for your design of Project #2.

Answer:



7. Write all the code for a function that attempts to convert an arbitrary type to a `std::string`. This should work for all the primitive C++ types, e.g., `int`, `double`, ... For which user defined types will your function succeed? How will your function fail to achieve its purpose?

Answer:

This function compiles and runs for any type that supplies an insertion operator for a `std::ostream`. If the insertion operator serializes the instance's state, then the function will behave as expected.

```
template<typename T>
std::string ToString(const T&t)
{
    std::ostringstream conv;
    if(typeid(t).name() == typeid(double).name())
        conv.precision(10);
    conv << t;
    return conv.str();
}
```