Translating UML Classes to C++

Guideline on Constructors vs. Setters

- If information is essential to the meaning of an object, it is better to pass it through the constructor than using a setter.
- Rationale: Using a setter, the object either has no meaning while awaiting for setting to occur, or the essential information has to have default values so that the object has meaning.
- Exception: Extremely large number of variables need to be passed, which would be confusing using the positional notation of a constructor.
- Exception: Two-way navigable associations; one object must be created before the other.

UML -> C++

Inheritance/Generalization

Guideline on Constructors vs. Setters

Shape s(p, s, c); // use nominal constructor
preferred over:
Shape s; // use default constructor
- - -
s.setPosition(p);
s.setSize(s);
s.setColor(c);

**Create a car of a specific make.**
Car(Car(const Car& c)); // constructor
- - -
Car(const string& make)
{ make_ = make; }

**Calls Chass is constructor**
UML -> C++:
One-way navigability

This UML says that one shape may have many connectors. One can get from a Connector to its Shape, but not vice-versa.

```cpp
class Shape {
public:
  // ...
private:
  // ...
};

class Connector {
public:
  // ...
private:
  Shape *shape; // Point to shape
};
```

This UML says that one shape may have many connectors. One can get from a Connector to its Shape, but not vice-versa.

```cpp
class Shape {
public:
  // ...
private:
  // ...
};

class Connector {
public:
  // ...
private:
  Shape *shape; // Point to shape
};
```

Disclaimer

- The C++ code examples are samples of what can be done.
- They are generally not the only way a specific type of association can be implemented.
- A specific tool (Rose, Rhapsody, etc.) may generate a specific type of implementation; selection from a menu of implementations might be possible.
- Use of the/a standard library, is possible and often advised for portability.

UML -> C++:
Multiple associations with different roles

```cpp
class Shape {
public:
  // ...
private:
  Connector *connector; // Point to connector
};

class Connector {
public:
  // ...
private:
  Shape *start; // Point to start
  Shape *end; // Point to end
};
```

UML -> C++:
1-1 association

```cpp
class Shape {
public:
  // ...
private:
  Connector *connector; // Point to connector
};

class Connector {
public:
  // ...
private:
  Shape *shape; // Point to shape
};
```

Generally choose one, not both, as a constructor for the other. Each calling the other could be a problem.