Design and Design Patterns

Design Patterns

Design Patterns use OO-principles to solve common problems.

Singleton Pattern

- Problem: Ensure a class has only one instance and provide a global point of access to that instance.

Singleton Class

```
class Singleton
{
public:
        static Singleton* Instance();
private:
        static Singleton* theSingletonInstance;
        Singleton() {};
        ~Singleton() {};
};

Singleton::Singleton* theSingletonInstance = NULL;
```

Instance Implementation

```
Singleton* Instance()
{
        if (theSingletonInstance == NULL)
                theSingletonInstance = new Singleton;
        return theSingletonInstance;
}
```

Access

```
Singleton* ptrTheSingleton = Singleton::Instance;
```

Example

```
class Ball
{
public:
        static Ball* theBall();
private:
        Sphere   theSphere;
        Ball() {};
        ~Ball() {};
};

Ball::Ball* theBall = NULL;
```

GRASP

General Responsibility Assignment Software Patterns

GRASP

- Information Expert
- Creator
- High Cohesion
- Low Coupling
- Controller

Information Expert

- Problem: What is a general principle of assigning responsibility to objects?
- Solution: Assign a responsibility to the information expert -- the class that has the information necessary to fulfill the responsibility.

Creator

- Problem: Who should be responsible for creating a new instance of some class?
- Solution: Assign Class B the responsibility to create an instance of class A if one or more of the following are true:
  - B aggregates A objects
  - B contains A objects
  - B records instances of A objects
  - B closely uses A objects
  - B has the initializing data that will be passed to A when it is created (B is an expert!)

Low Coupling

- Problem: How can design support low dependency, low change impact, and increased reuse?
- Solution: Assign responsibilities so coupling remains low.

High Cohesion

- Problem: How can design keep complexity manageable?
- Solution: Assign responsibility so that cohesion remains high.

Controller

- Problem: Who should be responsible for handling an input event?
- Solution: A class that
  - Represents the overall system, device, or subsystem
  - or, Manages a use case scenario within which the system event occurs

Triangle World Revisited

- Analyze your design wrt to GRASP.
- Re-design as necessary.
- Build sequence diagrams.