Class Design Principles Exercise Sheet

Problem 1

Imagine we are writing a small piece of software to draw various geometric objects.

Listing 1: **The Square/Circle Problem**

```
//--shape.hh--------------------------
enum ShapeType {circle, square};

struct Shape
{
   ShapeType itsType;
};

//--circle.hh--------------------------
struct Circle
{
   ShapeType itsType;
   double itsRadius;
   Point itsCenter;
};

void DrawCircle(Circle*);

//--square.hh--------------------------
struct Square
{
   ShapeType itsType;
   double itsSide;
   Point itsTopLeft;
};

void DrawSquare(Square*);

//--DrawAllShapes.cc--------------------------
void DrawAllShapes(Shape* list[], int n)
{
   int i;
   for (i=0; i<n; i++){
     Shape* s = list[i];
     switch (s->itsType)
     {
     case square:
       DrawSquare((struct Square*)s);
     break;
     case circle:
       DrawCircle((struct Circle*)s);
     break;
     }
   }
}
```

a) How many responsibilities has `DrawAllShapes` in Listing 1?

b) We are adding a new class `Triangle` and we want it to be drawn as well. How does Listing 1 adapt to this?

Problem 2

Assume 2 classes representing 2 related geometric entities.

Listing 2: **The Square/Rectangle Problem**

```
//--Rectangle.hh-----------------------------------------------
enum GeoType { Rectangle, Square };

class Rectangle
{
  public:
    virtual void SetWidth(double w)  {itsWidth=w;}
    virtual void SetHeight(double h) {itsHeight=h;}
    double       GetHeight() const   {return itsHeight;}
    double       GetWidth() const    {return itsWidth;}
    GeoType itsType;
  private:
    double itsHeight;
    double itsWidth;

};

//--Square.hh-----------------------------------------------
class Square : public Rectangle
{
  public:
  virtual void SetWidth(double w);
  virtual void SetHeight(double h);
};

void Square::SetWidth(double w)
{
  Rectangle::SetWidth(w);
  Rectangle::SetHeight(w);
}

void Square::SetHeight(double h)
{
  Rectangle::SetHeight(h);
  Rectangle::SetWidth(h);
}
```

Consider the following use of Rectangle and Square:

Listing 3: Using Square and Rectangle

```
  void g(Rectangle& r)
  {
    r.SetWidth(5);
    r.SetHeight(4);
    assert(r.GetWidth() * r.GetHeight()) == 20);
  }
```

a) What will happen if Listing 3 is called with a `Square` or a `Rectangle` object?

b) Given the design in Listing 2, what counter-measures are necessary to make Listing 3 work?

Problem 3

Given the following `Lamp` class definition:

Listing 4: A Lamp class

```
class Lamp
{
  public:
    void TurnOn();
    void TurnOff();
};
```

a) Write or sketch a `Button` class that turns `Lamp` on and off!

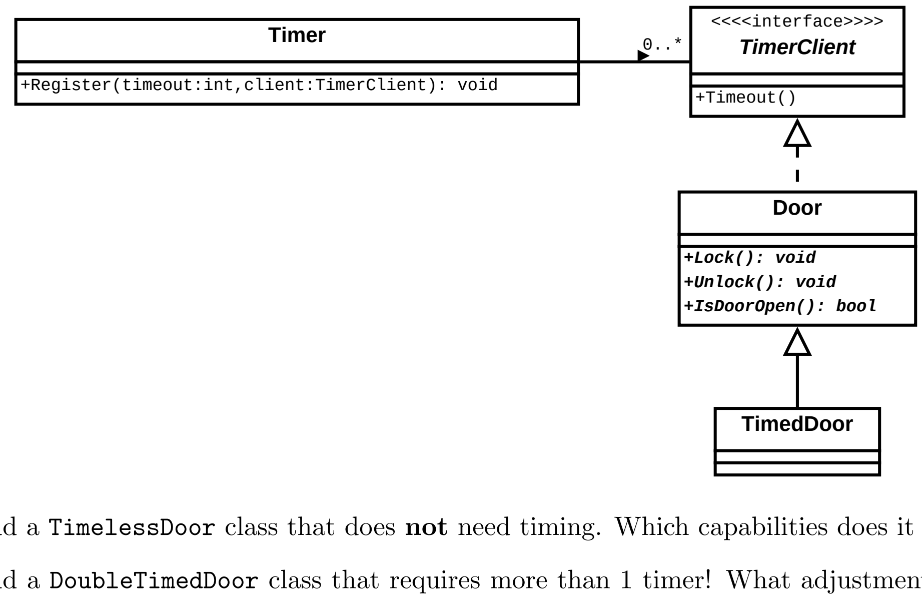
Problem 4

You are asked to code a security door device that will alarm once an attach door is kept open too long. You come up with the following classes: In order to accomplish your task, you choose the

Listing 5: A Door class

```
class Door
{
  public:
    virtual void Lock()          = 0;
    virtual void Unlock()        = 0;
    virtual bool IsDoorOpen()    = 0;
};

class Timer
{
  public:
     void Register(const int\& timeout, TimerClient* client);

};

class TimerClient
{
  public:
     virtual void TimeOut() = 0;

};
```

following software setup.



a) Add a `TimelessDoor` class that does **not** need timing. Which capabilities does it have?

b) Add a `DoubleTimedDoor` class that requires more than 1 timer! What adjustments do you need to make besides adding a derived class of `Door`?

Disclaimer

Source code snippets from exercises **1.a)**, **2.**, **3.**, **4.** and **5.** were adapted from the book:

author	Martin, Robert C. and Newkirk, James W. and Koss, Robert S.
title	Agile Software Development
publisher	Prentice Hall
year	2003
note	http://www.objectmentor.com/resources/publishedArticles.html