C++ Primer Plus, 5^{th} ***Edition*** by Stephen Prata
Chapter 14: Reusing Code in C++
Review Questions

1. For each of the following sets of classes, indicate whether public or private derivation is more appropriate for Column B:

A	B
class Bear	class PolarBear
class Kitchen	class Home
class Person	class Programmer
class Person	class HorseandJockey
class Person, class Automobile	class Driver

- *The PolarBear class should be derived publically since a polar bear is a kind of bear.*
- *The Home class should derive the Kitchen class privately since a home has a kitchen.*
- *The Programmer class should use public derivation since a programmer is a person.*
- *The HorseandJockey class should use private derivation since a horse and jockey have a person.*
- *The Driver class should use public derivation to inherit the Person class since a driver is a person. This class should also use private derivation to inherit the Automobile class since a driver has an automobile.*

2. Suppose you have the following definitions:

```
class Frabjous {
private:
    char fab[20];
public:
    Frabjous(const char * s = "C++") : fab(s) { }
    virtual void tell () { cout << fab; }
};

class Gloam {
private:
    int glip;
    Frabjous fb;
public:
    Gloam(int g = 0, const char * s = "C++");
    Gloam(int g, const Frabjous & f);
    void tell();
};
```

Given that `Gloam` version of `tell()` should display the values of `glip` and `fb`, provide definitions for the three `Gloam` methods.

See the following code:

```
Gloam::Gloam(int g, const char * s) : fb(s), glip(g)
{
}

Gloam::Gloam(int g, const Frabjous & f) : fb(f), glip(g)
{
}

void Gloam::tell()
{
   std::cout << "fb = ";
   fb.tell();
   std::cout << std::endl;
   std::cout << "glip = " << glip << std::endl;
}
```

3. Suppose you have the following definitions:

```
class Frabjous {
private:
    char fab[20];
public:
    Frabjous(const char * s = "C++") : fab(s) { }
    virtual void tell () { cout << fab; }
};

class Gloam : private Frabjous {
private:
    int glip;
public:
    Gloam(int g = 0, const char * s = "C++");
    Gloam(int g, const Frabjous & f);
    void tell();
};
```

Given that `Gloam` version of `tell()` should display the values of `glip` and `fb`, provide definitions for the three `Gloam` methods.

See the following code:

```
Gloam::Gloam(int g, const char * s) : Frabjous(s), glip(g)
{
}

Gloam::Gloam(int g, const Frabjous & f) : Frabjous(f), glip(g)
{
}

void Gloam::tell()
{
   std::cout << "fb = ";
   Frabjous::tell();
   std::cout << std::endl;
   std::cout << "glip = " << glip << std::endl;
}
```

4. Suppose you have the following definition, based on the `Stack` template of Listing 14.13 and the `Worker` class of Listing 14.10:

   ```
   Stack<Worker *> sw;
   ```

   Write out the class declaration that will be generated. Just do the class declaration, not the non-inline class methods.

   See the following code:

   ```
   class Stack
   {
   private:
      enum {MAX = 10};
      Worker * items[MAX];
      int top;
   public:
      Stack();
      bool isempty();
      bool isfull();
      bool push(const Worker * & item);
      bool pop(Worker * & item);
   };
   ```

5. Use the template definitions in this chapter to define the following:
   - An array of `string` objects
   - A stack of arrays of `double`
   - An array of stacks of pointers to `Worker` objects

   How many template class definitions are produced in Listing 14.18?

   - `valarray<string> arr;`
   - `stack< valarray<double> > stk;`
   - `valarray< stack<Worker *> > arr;`

   Four template class definitions are produced in Listing 14.18.

6. Describe the differences between virtual and nonvirtual base classes.

   The difference arrises when a derived object inherits two or more classes that were each derived from the same base object. If we have non-virtual base classes, then the final object inherits as many copies of the original base class as it inherits classes that were derived from the original base class. If we have virtual base classes, then the final derived class inherits a single copy of the original base class that is shared between all virtual base classes.