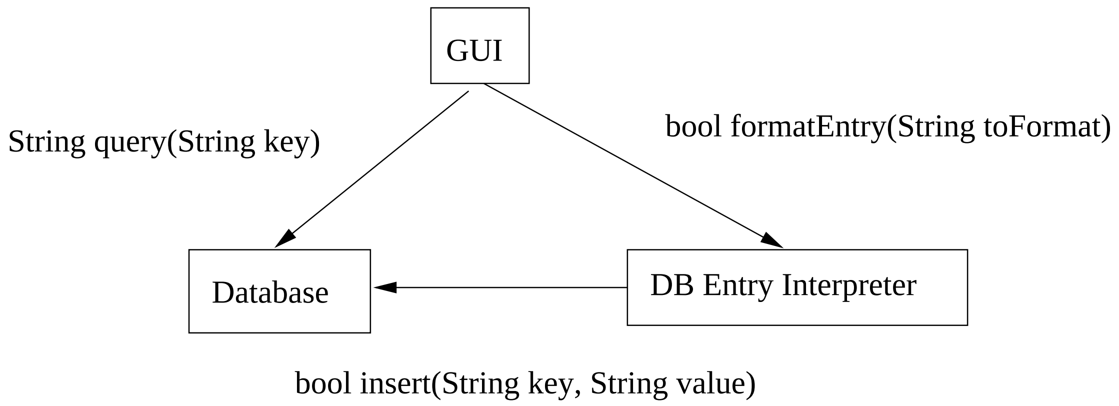
CS 190 Levelization Assignment

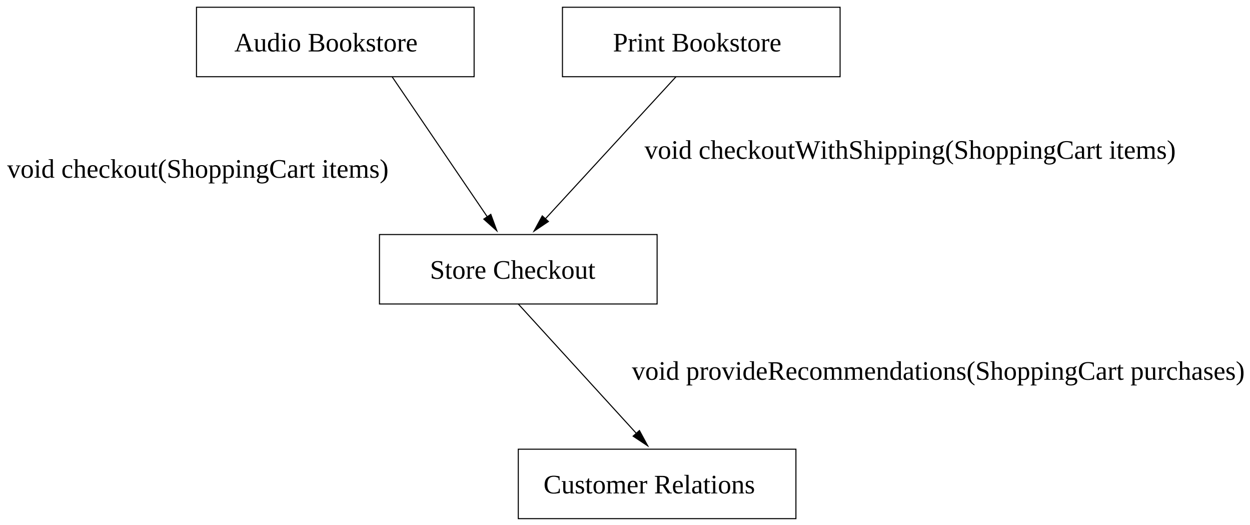
For problems one through six, levelize (i.e., assign level numbers to the components and write them on the original diagram) the original diagram and then consider how to "improve" it by removing cyclic dependencies, making it shorter, making it wider, or a combination of the three. Your solutions should be new levelized diagrams (once again, with level numbers written on them). Hand-written solutions are fine!

Improve the following design using escalation.


GUI
String query(String key)
bool formatEntry(String toFormat)
Database
DB Entry Interpreter
bool insert(String key, String value)

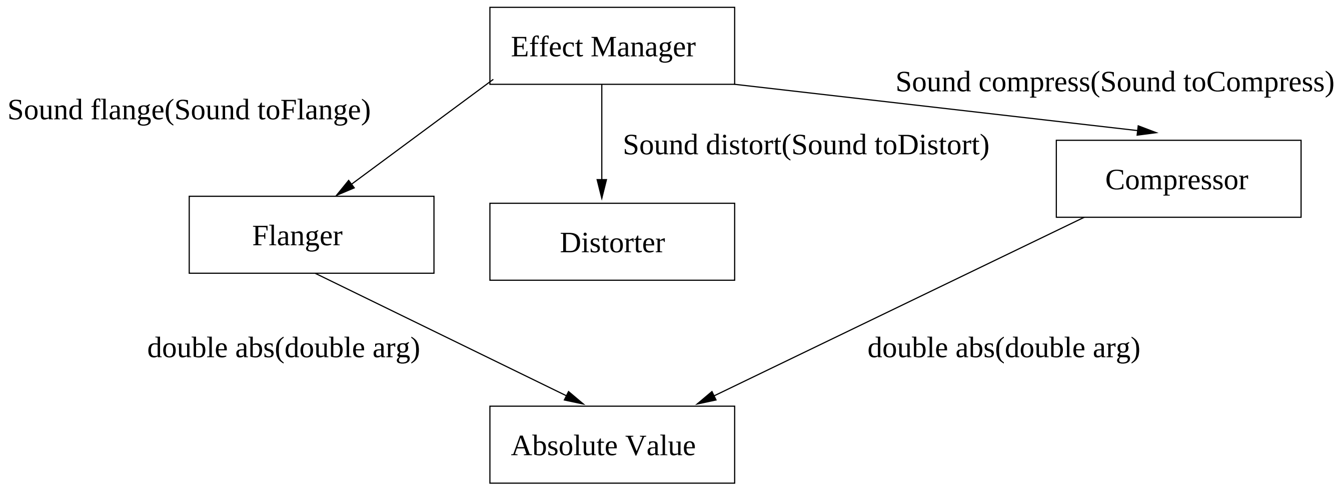
This diagram represents a simple database application. The `GUI` is, well, a GUI. It sends information from the user to the `DBEntryInterpreter` with a `formatEntry()` function (which returns `true` on success and `false` on failure). The `DBEntryInterpreter` then formats this data to insert into `Database` and calls `Database`'s `insert()` function. The GUI can also perform queries on `Database` by calling its `query()` function.

Improve the following design using demotion.


Audio Bookstore
Print Bookstore
void checkout(ShoppingCart items)
void checkoutWithShipping(ShoppingCart items)
Store Checkout
void provideRecommendations(ShoppingCart purchases)
Customer Relations

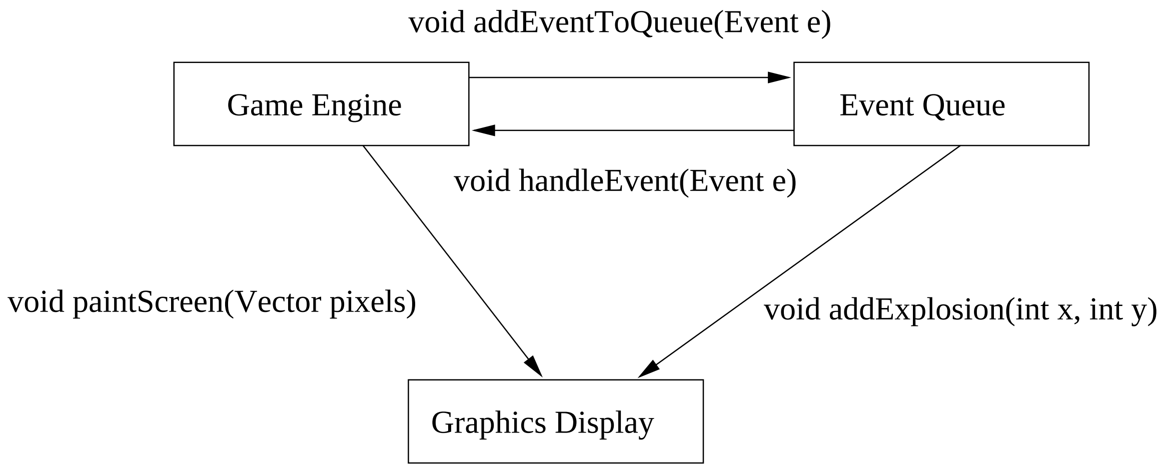
This diagram represents a very simple online store. The `AudioBookstore` component manages sales of ”podcast” books – audio downloads – whereas the `PrintBookstore` component sells physical books from a warehouse stock. Both of these components call the `StoreCheckout` component (with the `checkout()` and `checkoutWithoutShipping()` functions) when a user decides to make a purchase. `StoreCheckout` handles both shipping and credit card transactions. When these transactions are done, `StoreCheckout` calls the the `provideRecommendations()` function on `CustomerRelations`, providing the user with a list of suggested future purchases based on their transaction.

Improve the following design using redundancy.



This diagram represents the portion of a wave file editor that applies various effects to a sound wave. The `EffectManager` takes the choices the user makes and performs the appropriate effect on the sound by passing the selected audio to `Flanger`, `Compressor`, or `Distorter`. Two of the three effects require absolute value, so it's been placed in a separate component (aptly named `AbsoluteValue`), which has an `abs()` call.

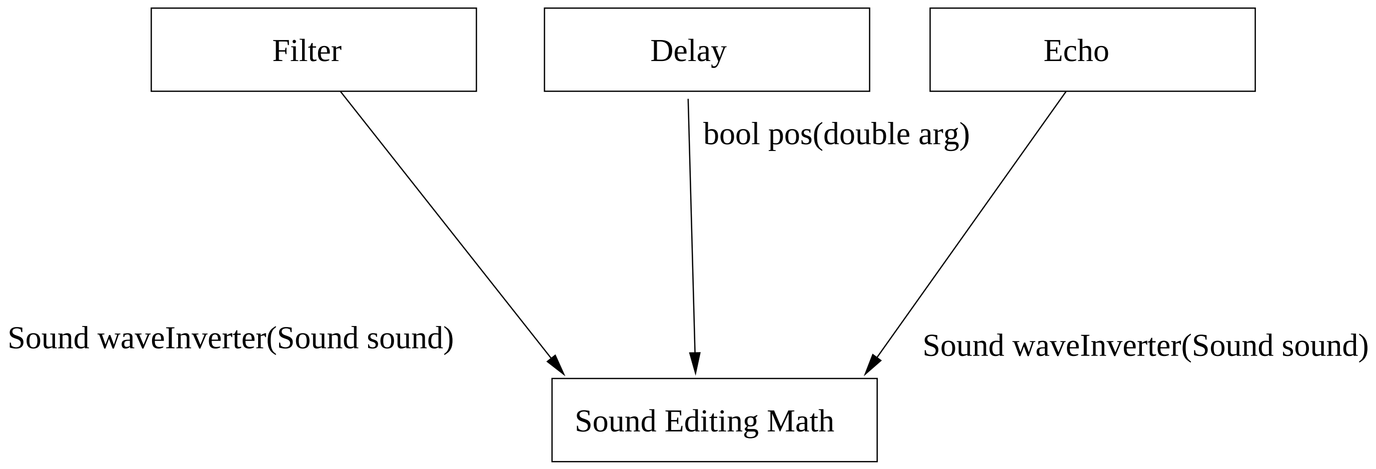
Improve the following design with a manager class.


void addEventToQueue(Event e)
Game Engine
Event Queue
void handleEvent(Event e)
void paintScreen(Vector pixels)
void addExplosion(int x, int y)
Graphics Display

This is a diagram for part of a game. The `GameEngine` is a continuous loop that regularly updates the display (using `GraphicsDisplay`'s `paintScreen()`

function), controls enemy movement, etc. Meanwhile, the `EventQueue` handles collisions between player and enemy characters and thus occasionally needs to draw an explosion on the screen (also using the `GraphicsDisplay`. Occasionally, an event occuring in the `EventQueue` will have an effect on the `GameEngine`'s operations, and the game engine will occasionally have to add an event to the queue, so they must communicate with each other as shown in the diagram.

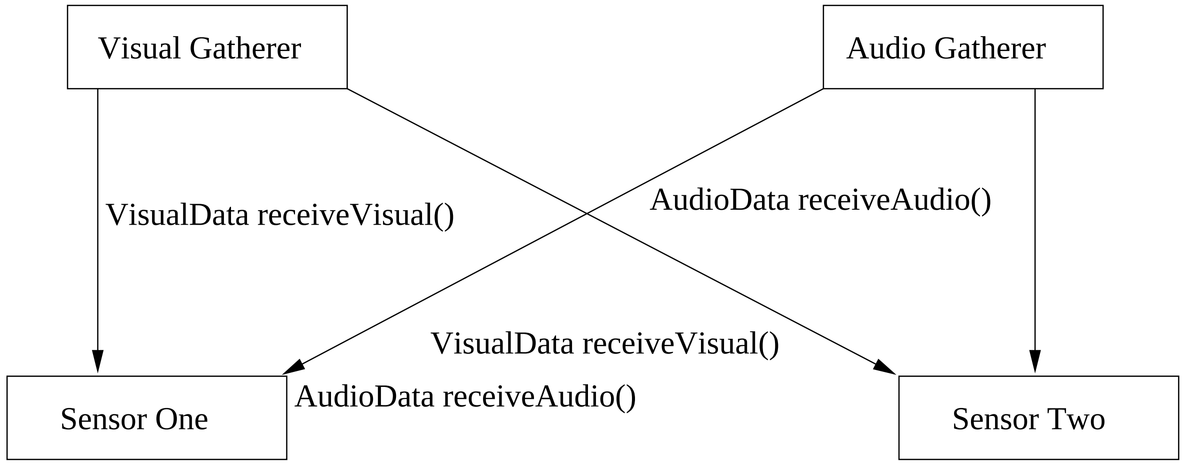
Improve the following design by refactoring.



Filter
Delay
Echo
bool pos(double arg)
Sound waveInverter(Sound sound)
Sound waveInverter(Sound sound)
Sound Editing Math

Consider the effects-application portion of a wave file editor again. Now we have three different effects (`Filter`, `Delay`, `Echo`) that use a variety of mathematical algorithms to manipulate the sound, all of which are performed by a Math component (`SoundEditingMath`) by calling `pos()` or `waveInverter()`.

Improve the following design by escalating encapsulation.



Consider a sensor network with three different types of sensors; each type of sensor is controlled by a software component written specifically for that sensor (`SensorOne`, `SensorTwo`). There are two separate components that grab and analyze data from the sensors independently: `AudioGatherer` and `VisualGatherer`.

Improve the .h file using an opaque pointer.

For this problem, you need not provide a diagram; provide a modified .h file for this odd software system that uses an opaque pointer. Explain briefly how it makes the design more independently testable.

```
class RobotMaid {
    MaidController controller;
    MotherBrain mother;
    // ...
    public:
        Maid(MaidController control, MotherBrain mother);
        int getControllerID(); // extracts info from this Maid's controller
                               // and returns it. Usually called by a
                               // MotherBrain.
        // ...
};
```