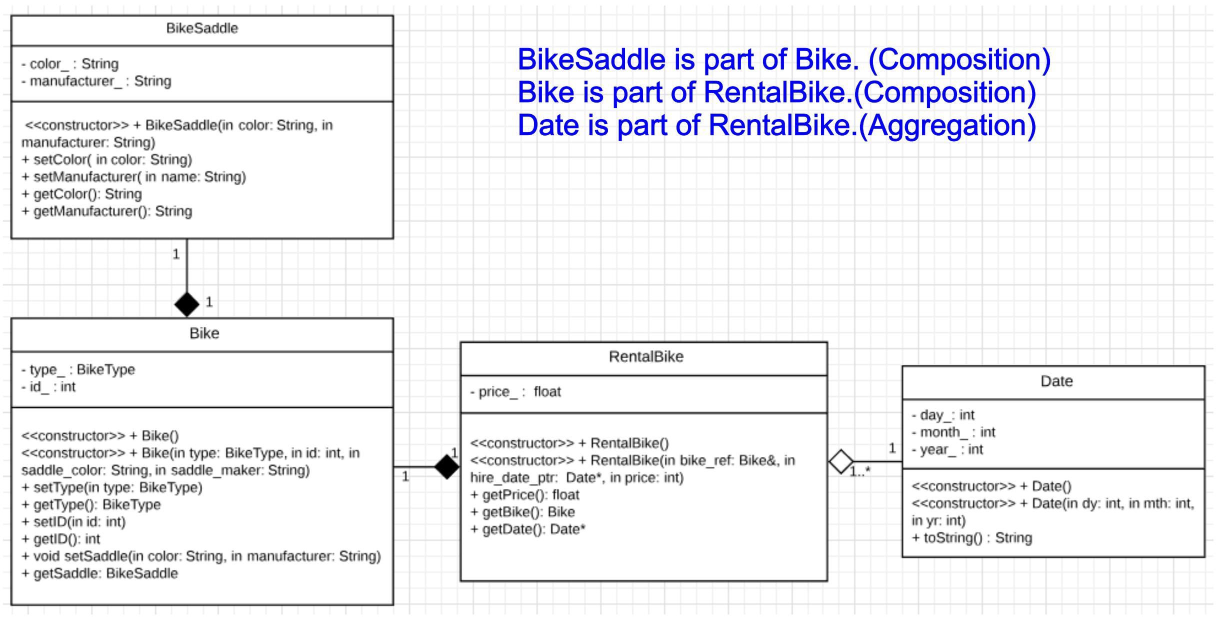
Computing 2 Homework Problems

Problem Set 2 (Separating Interfaces from Implementations, Composition and Associations)

In this homework problem, we will continue to work with our `Bike` class that has a `BikeSaddle`. To this end, we define the class `RentalBike` and the class `Date.` Class `Bike` is composed with `RentalBike,` while class `Date` is aggregated with `RentalBike.` In addition, a member variable `price_` of type `float` was added to `RentalBike.` The relationships between all of the classes are illustrated in the UML class diagram below.


BikeSaddle
- color_ : String
- manufacturer_ : String
<<constructor>> + BikeSaddle(in color: String, in manufacturer: String)
+ setColor(in color: String)
+ setManufacturer(in name: String)
+ getColor(): String
+ getManufacturer(): String
1
1
Bike
- type_ : BikeType
- id_ : int
<<constructor>> + Bike()
<<constructor>> + Bike(in type: BikeType, in id: int, in saddle_color: String, in saddle_maker: String)
+ setType(in type: BikeType)
+ getType(): BikeType
+ setID(in id: int)
+ getID(): int
+ void setSaddle(in color: String, in manufacturer: String)
+ getSaddle: BikeSaddle
1
1
RentalBike
- price_ : float
<<constructor>> + RentalBike()
<<constructor>> + RentalBike(in bike_ref: Bike&, in hire_date_ptr: Date*, in price: int)
+ getPrice(): float
+ getBike(): Bike
+ getDate(): Date*
1..*
1
Date
- day_: int
- month_ : int
- year_ : int
<<constructor>> + Date()
<<constructor>> + Date(in dy: int, in mth: int, in yr: int)
+ toString() : String
BikeSaddle is part of Bike. (Composition)
Bike is part of RentalBike.(Composition)
Date is part of RentalBike.(Aggregation)

For a composition relationship, the part must be a constitute part of the whole object. For example, a heart is a part of a person's body. The part in a composition can only be part of one object at a time. A heart is part of one person's body. It cannot be part of someone else's body at the same time. In a composition relationship, the whole object is responsible for the existence of the part. If the whole object is deleted, the part is deleted with it. Compositions are used to create complex objects from simpler ones. *Composition is implemented in C++ as a class member variable.* composition

Similar to a composition, an aggregation is still a part-whole relationship, where the parts are contained within the whole. However, unlike a composition, parts can belong to more than one object at a time, and the whole object is not responsible for the existence and lifespan of the parts. When an aggregation is created, the aggregation is not responsible for creating the parts. When an aggregation is destroyed, the aggregation is not responsible for destroying the parts. *Aggregation is implemented as a member pointer variable*. aggregation

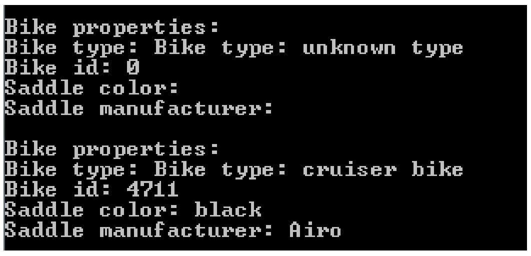
In general, the variables implementing composition and aggregation are not included in the class diagram of the parent class. It is implicitly understood that the

- *UML composition symbol is translated into C++ as a member variable,*
- *UML aggregation symbol is translated into C++ as a pointer variable*

It is left to the programmer to choose appropriate variable names to represent these relationships. This is also why we no longer mention a `BikeSaddle` variable as part of the `Bike` class.

Your tasks are:

1. Finish up the second part of the previous homework. This is, separate the interfaces from the implementations by writing the following files:
   - o bikesaddle.h (header file for class `BikeSaddle`)
   - o bikesaddle.cpp (implementations of the member functions of class `BikeSaddle`)
   - o bike.h (header file for class `Bike`)
   - o bike.cpp (implementations of the member functions of class `Bike`)
2. Also write a file main.cpp implementing the `main` function. Make sure that everything runs and produces the same output as before. This completes the first part of the homework.



```
Bike properties:
Bike type: Bike type: unknown type
Bike id: 0
Saddle color:
Saddle manufacturer:

Bike properties:
Bike type: Bike type: cruiser bike
Bike id: 4711
Saddle color: black
Saddle manufacturer: Airo
```

3. For the second part of the homework, implement the class `Date` according to the UML diagram first. Write the header file date.h and the source file date.cpp.
4. Create the class `RentalBike` according to the UML diagram. Write the header file rentalbike.h and the source file rentalbike.cpp.
5. Overload the `display(Bike bike)` function of the Bike class to show the properties of the RentalBike class, see below.
6. Modify your main function such that you can test your new classes `Date` and `RentalBike`.

Additional Information

Here is the definition of the function `void display(RentalBike rental_bike)` which you need to generate the output in the `main` function:

```
void display(RentalBike rental_bike)
{
      cout << "\nRental Bike properties: " << endl;
      cout << "Bike type: ";
      switch( //your code here )
      {
         case kUnknown:      cout << "unknown type" << endl; break;
         case kRoadBike:     cout << "road bike" << endl; break;
         case kTouringBike:  cout << "touring bike" << endl; break;
         case kMountainBike: cout << "mountain bike" << endl; break;
         case kCruiser:      cout << "cruiser bike" << endl; break;
         case kCityBike:     cout << "city bike" << endl; break;
         case kBmxBike:      cout << "BMX bike" << endl; break;
         case kFoldingBike:  cout << "folding bike" << endl; break;
         case kTandem:       cout << "folding bike" << endl; break;
         case kTricycle:     cout << "tricycle" << endl; break;
         default:
            cout << "no valid bike type!" << endl;
      }
      cout << "Bike id: " << //your code here () << endl;
      cout << "Saddle color: " << //your code here << endl;
      cout << "Saddle manufacturer: " << //your code here << endl;
      cout << "Hire Date: " << //your code here << endl;
      cout << "Rental Price: " << //your code here << endl;
      return;
}
```

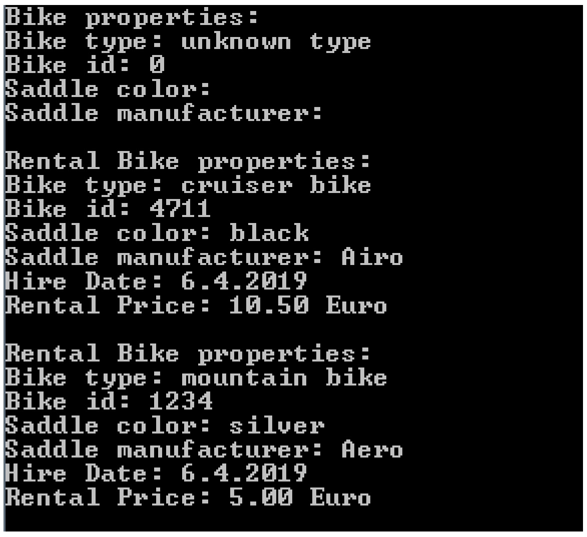
Here is the definition of the function `toString()` of the `Date` class. It requires the `sstream` header (`#include <sstream>`):

```
string Date::toString() const
{
      ostringstream output;
      output << day_ << '.' << month_ << '.' << year_;
      return output.str();
}
```

And here is a suggestion for the `main()` function.

```
int main()
{
   Date* your_hire_date_ptr = new Date(6,4,2019);
   //cout << "Bike Hire Date: " << your_hire_date_ptr->toString() << endl;

   Bike my_bike;
   display(my_bike);

   Bike bike_1(kCruiser, 4711, "black", "Airo");
   //display(bike_1);
   RentalBike rental_bike_1(bike_1, your_hire_date_ptr, 10);
   display(rental_bike_1);


   Bike bike_2(kMountainBike, 1234, "silver", "Aero");
   RentalBike rental_bike_2(bike_2, your_hire_date_ptr, 5);
   display(rental_bike_2);

   delete your_hire_date_ptr;

   return 0;
}
```

It should produce the following output.



```
Bike properties:
Bike type: unknown type
Bike id: 0
Saddle color:
Saddle manufacturer:

Rental Bike properties:
Bike type: cruiser bike
Bike id: 4711
Saddle color: black
Saddle manufacturer: Airo
Hire Date: 6.4.2019
Rental Price: 10.50 Euro

Rental Bike properties:
Bike type: mountain bike
Bike id: 1234
Saddle color: silver
Saddle manufacturer: Aero
Hire Date: 6.4.2019
Rental Price: 5.00 Euro
```