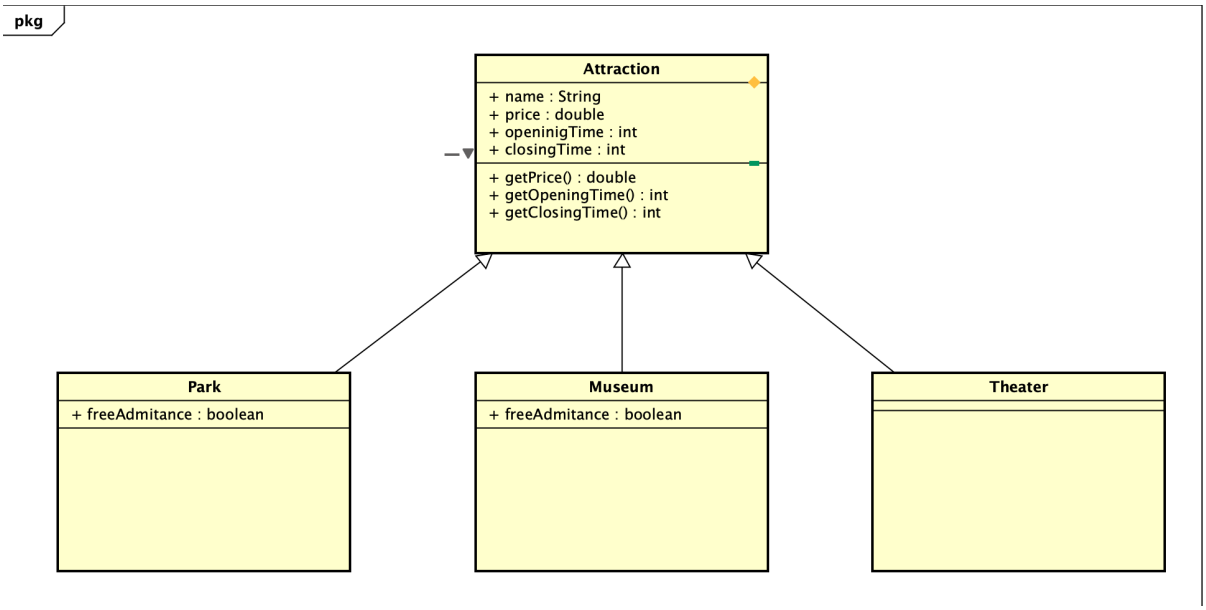
Table of Contents

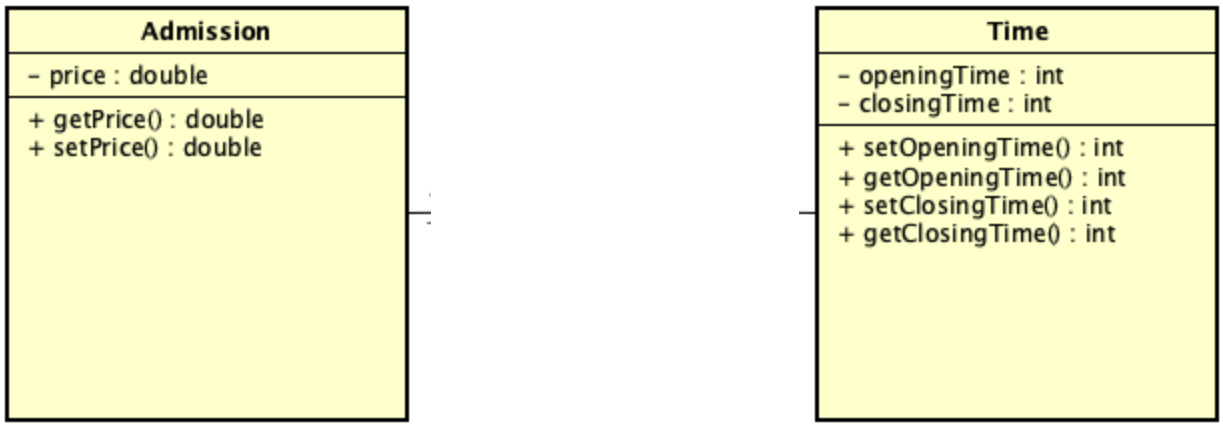
Question - 01

If we get a simple approach for this question the whole thing can be resolved by using a simple class file name as attraction. But it will eventually lead us to the major questions about scalability and flexibility. When we are designing a system like this it will need to develop in a manner that can be easy to add features without breaking anything and also it should be able to scale at any time. As for the next step we can divide the main attractions into subclasses such as park, museum and theater and inherit it from the main attraction class as shown below.

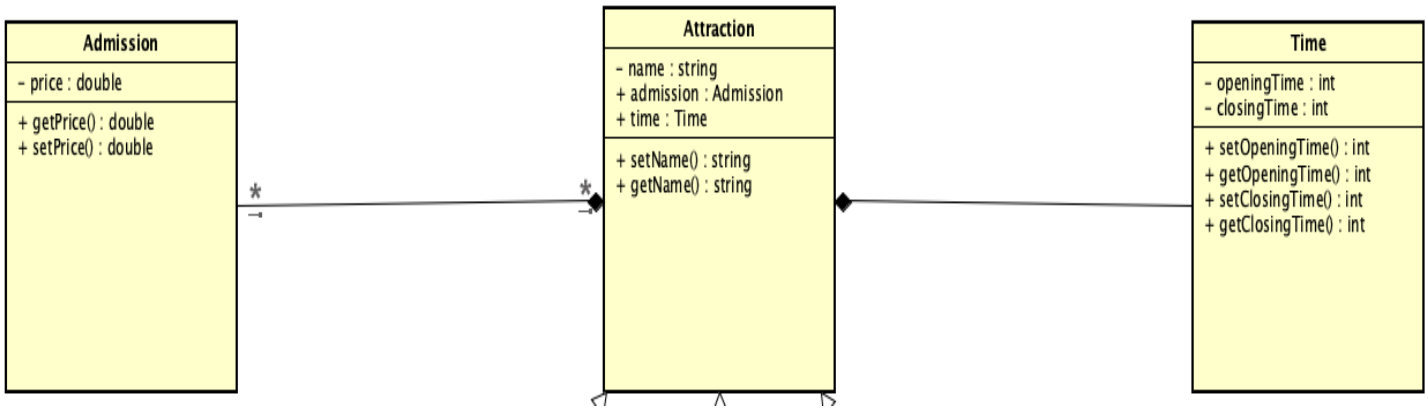


Here key attributes and methods of the attraction class will be inherited to all the subclasses. If there is a specific change in prices we can override it and use it. But the issue here is there is a ton of code duplication and if there is a requirement to add more attractions with specific pricing instructions then we have to override all of those classes one by one. It's not an ideal solution when the application starts to scale and there is no uniform manner in the code when we code like that.

Since inheritance is not an ideal solution here we have to use design principles and think through it. ***"Identify the aspects of your application that vary and separate them from what stays the same".*** As this design principle says we have to separate the components that constantly change from the components that stay the same. In this particular application components that always vary are the pricing and time. So we can split them into two separate classes. We can set getters and setters for interacting with the variables. Simply what we have done here is take the components that vary and **encapsulate** them.



Next we have to identify the relationship between Attraction, admission and time classes. We can not use inheritance. As we previously explained it will cause unnecessary code duplication and we can not achieve uniformity of the code that we are seeking for the pricing model. This will take us to the next design principle we will use in this application. ***"Favor composition over inheritance".*** We will use time and price classes as a composition of the attraction class.

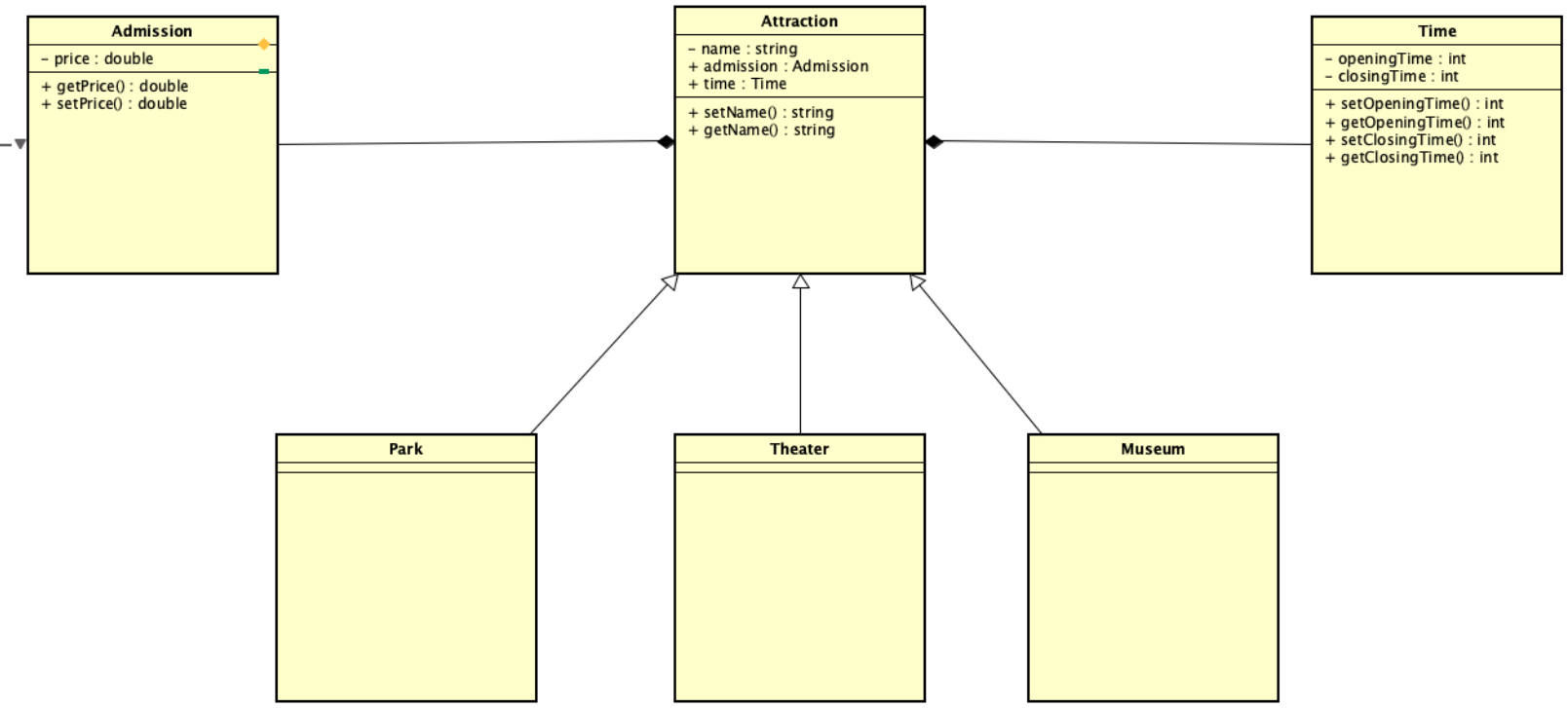


Instead of inheriting their behavior, the attractions get their behavior by being composed with the right behavior object. Creating a system using composition will give us the following benefits,

- Gives a lot more flexibility
- Encapsulate a family of algorithms into their own set of classes.
- Ability to change the behavior at the run time.

In the final design Admission and time class will be composition of the attraction class and park, theater and museum class will be inherited from the attraction class. Main reason for inheritance is of attraction types is we can create new feature for the specific attractions

Question - 02



Park ,museum and theater classes inherit from the attraction class. They can be use if there is any attributes or functions that specific for each attraction type.

Question - 03

Following is the code snippets how each class is created according to the class diagram. Each class is split into header and implementation classes to follow the good coding practices in C++.

Admission class

The following code snippet is the representation of the admission class structure. The admission class has the private price attribute of double. The getPrice() method is a public getter that returns the current value of the price attribute. setPrice() method is a public setter that takes a double argument as the new price. It performs basic validation to ensure the price is not negative. If the price is valid, it updates the price attribute. If the getPrice() is zero it is considered as free of admission.

admission.h

```
#ifndef ADMISSION_H
#define ADMISSION_H

class Admission {
public:
  // Getter and setter methods
  double getPrice() const;
  void setPrice(double newPrice);

  // Optional constructor
  Admission(double initialPrice = 0.0);

private:
  // Price attribute
  double price;
};

#endif
```

admission.cpp

```
#include "Admission.h"
```

```cpp
double Admission::getPrice() const {
  return price;
}

void Admission::setPrice(double newPrice) {
  if (newPrice < 0) {
    std::cerr << "Error: Price cannot be negative." << std::endl;
  } else {
    price = newPrice;
  }
}

Admission::Admission(double initialPrice) : price(initialPrice) {}
```

Time class

This code snippet represents the time class. openingTime_ and closingTime_ attributes as private members. Getter and setter methods for both attributes with basic validation on input values. An optional constructor that allows to specify both opening and closing times when creating a Time object.

time.h

```cpp
#ifndef TIME_H
#define TIME_H

class Time {
public:
  // Getter and setter methods
  int getOpeningTime() const;
  void setOpeningTime(int openingTime);

  int getClosingTime() const;
  void setClosingTime(int closingTime);

  // Optional constructor (assumes 24-hour format)
  Time(int openingTime = 0, int closingTime = 23);

private:
  // Time attributes
  int openingTime_;
```

```
  int closingTime_;
};

#endif
```

time.cpp

```
#include "Time.h"

int Time::getOpeningTime() const {
  return openingTime_;
}

void Time::setOpeningTime(int openingTime) {
  if (openingTime < 0 || openingTime > 23) {
    std::cerr << "Error: Opening time must be between 0 and 23." << std::endl;
  } else {
    openingTime_ = openingTime;
  }
}

int Time::getClosingTime() const {
  return closingTime_;
}

void Time::setClosingTime(int closingTime) {
  if (closingTime < 0 || closingTime > 23) {
    std::cerr << "Error: Closing time must be between 0 and 23." << std::endl;
  } else if (closingTime < openingTime_) {
    std::cerr << "Error: Closing time cannot be before opening time." << std::endl;
  } else {
    closingTime_ = closingTime;
  }
}

Time::Time(int openingTime, int closingTime) : openingTime_(openingTime),
closingTime_(closingTime) {}
```

Attraction class

The Attraction class has a private data member called name_ to store the attraction's name and It composes both Time and Admission objects as private data members. Public getter and setter methods are provided for

accessing the name, time, and admission components. from getTime() and getAdmission() methods offer both const and non-const versions, allowing for both read-only access and modification depending on the need. The constructor allows to initialize an Attraction object with a name, time, and admission details.

Attraction.h

```
#ifndef ATTRACTION_H
#define ATTRACTION_H

#include "Time.h"
#include "Admission.h"

class Attraction {
public:
  // Getter and setter methods for name
  std::string getName() const;
  void setName(const std::string& name);

  // Getter and setter methods for Time component
  const Time& getTime() const;
  Time& getTime();

  // Getter and setter methods for Admission component
  const Admission& getAdmission() const;
  Admission& getAdmission();

 Attraction(const std::string& name, const Time& time, const Admission&
admission);

private:
  // Attraction name
  std::string name_;

  // Time component (composition)
  Time time_;

  // Admission component (composition)
  Admission admission_;
};
```

```
#endif
```

Atraction.cpp

```
#include "Attraction.h"

std::string Attraction::getName() const {
  return name_;
}

void Attraction::setName(const std::string& name) {
  name_ = name;
}

const Time& Attraction::getTime() const {
  return time_;
}

Time& Attraction::getTime() {
  return time_;
}

const Admission& Attraction::getAdmission() const {
  return admission_;
}

Admission& Attraction::getAdmission() {
  return admission_;
}

Attraction::Attraction(const std::string& name, const Time& time, const
Admission& admission) :
  name_(name), time_(time), admission_(admission) {}
```

Park class

This class is inherited from the attraction class.

park.h

```
#ifndef PARK_H
#define PARK_H

#include <string>
#include "Attraction.h"

class Park : public Attraction {
public:
   // Optional: Constructor that takes additional park-specific arguments
   Park(const std::string& name, const Time& time, const Admission& admission);

private:
   // Park-specific data members

};

#endif
```

park.m

```
#include "Park.h"

Park::Park(const std::string& name, const Time& time, const Admission&
admission)
       : Attraction(name, time, admission){}
```

Theater class

Theater.h

```
#ifndef THEATRE_H
#define THEATRE_H

#include <string>
#include "Attraction.h"

class Theatre : public Attraction {
public:
   // Optional: Constructor that takes additional park-specific arguments
   Theatre(const std::string& name, const Time& time, const Admission&
admission);
```

```
private:
   // Park-specific data members

};

#endif
```

Theater.cpp

```
#include "Theatre.h"

Theatre::Theatre(const std::string& name, const Time& time, const Admission&
admission)
       : Attraction(name, time, admission){}
```

Museum class

museum.h

```
#ifndef MUSEUM_H
#define MUSEUM_H

#include <string>
#include "Attraction.h"

class Museum : public Attraction {
public:
   // Optional: Constructor that takes additional park-specific arguments
   Museum(const std::string& name, const Time& time, const Admission&
admission);

private:
   // Park-specific data members

};

#endif
```

museum.cpp

```
#include "Museum.h"

Museum::Museum(const std::string& name, const Time& time, const Admission&
admission)
       : Attraction(name, time, admission){}
```

Main.cpp

Main function can find which attractions are open after 1900 and cost less than £5.

```
#include <iostream>
#include <vector>
#include "Attraction.h"
#include "Park.h"
#include "Museum.h"
#include "Theatre.h"

using namespace std;

int main() {
   // Create some example attractions
   Park park1("Central Park", Time(10, 18), Admission(10.0));
   Museum museum1("National Museum", Time(9, 17), Admission(8.0));
   Theatre theater1("Grand Theater", Time(12, 22), Admission(15.0));
   Park park2("North Park", Time(7, 18), Admission(0.0));
   Museum museum2("Historic Museum", Time(9, 17), Admission(4.0));
   Theatre theater2("London Theater", Time(12, 22), Admission(25.0));

   // Store attractions in a vector
   vector<Attraction*> attractions;
   attractions.push_back(&park1);
   attractions.push_back(&museum1);
   attractions.push_back(&theater1);

   // Find attractions open after 1900 and costing less than £5
   cout << "Attractions open after 7 PM and costing less than £5:" << endl;
   for (const Attraction* attraction : attractions) {
       if (attraction->getTime().getClosingTime() > 19 &&
attraction->getAdmission().getPrice() < 5.0) {
```

```
            cout << "- " << attraction->getName() << endl;
        }
    }

    return 0;
}
```