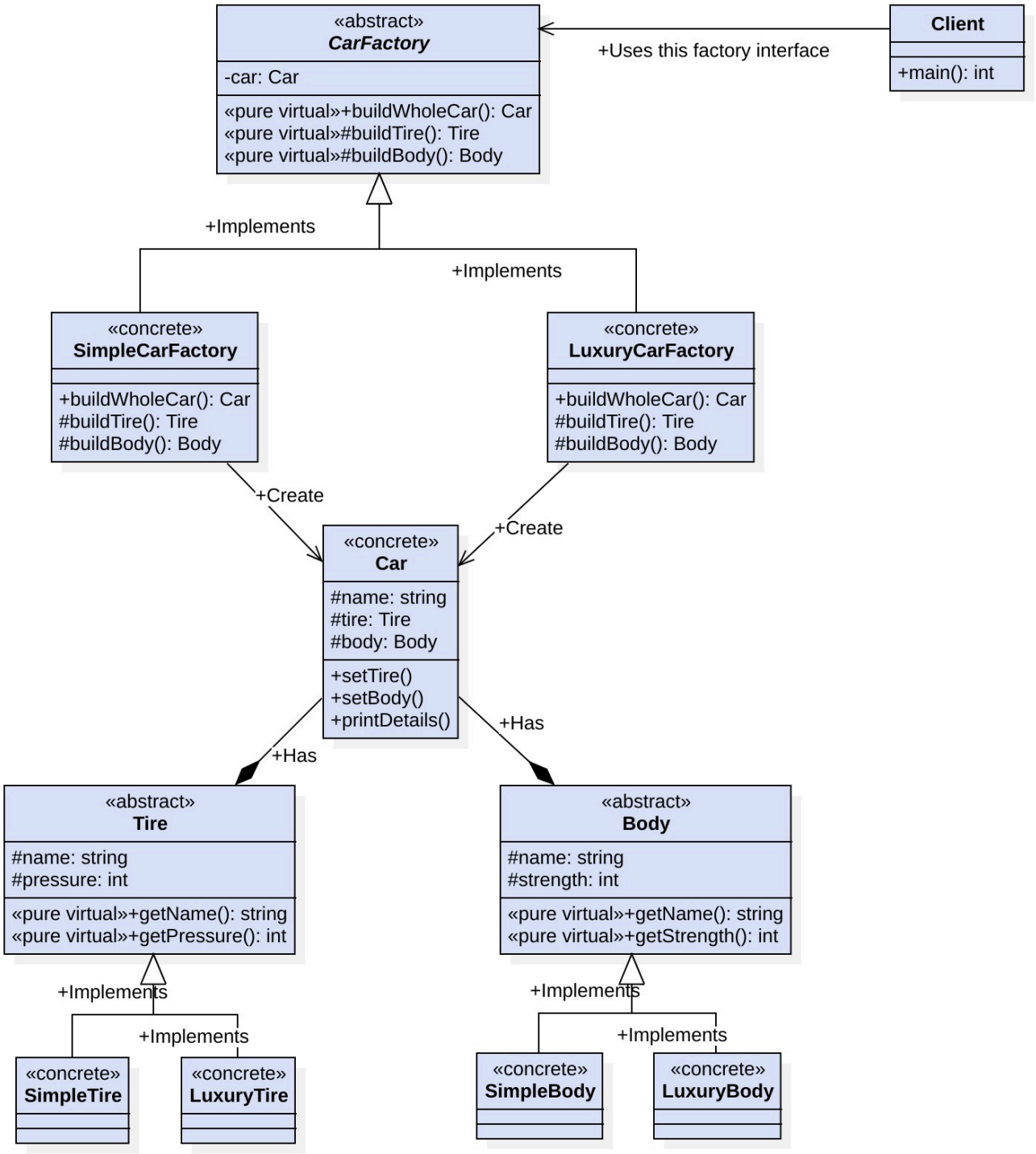
AFDP - Abstract Factory Design Pattern defines an abstract class for creating families of related objects without specifying their concrete sub-classes.

## How to implement AFDP



## Why & When to use AFDP

- You need the system to be independent of how objects are created, composed and represented
- You only want to expose interfaces, not implementations (e.g. proprietary library).
- The system needs to be configured with one of multiple families of objects.

## Reference Implementation

### src/AFDPDemo.cpp

```
#include "CarFactory.cpp"
using AFDP::Car;
using AFDP::CarFactory;
using AFDP::LuxuryCarFactory;
using AFDP::SimpleCarFactory;

// INFO: Macros contorl type of factory (uncomment to set type)
// #define SIMPLE_CAR 1
#define LUXURY_CAR 1

int main(int argc, char *argv[]) {
#ifdef SIMPLE_CAR

  CarFactory *factory = new SimpleCarFactory();

#elif LUXURY_CAR
  CarFactory *factory = new LuxuryCarFactory();

#endif // DEBUG

  Car *car = factory->buildWholeCar();
  car->printDetails();

  return 0;
}
```

### src/CarFactory.cpp

```
#include "Car.cpp"
using AFDP::LuxuryBody;
```

```
using AFDP::LuxuryTire;
using AFDP::SimpleBody;
using AFDP::SimpleTire;

namespace AFDP {

class CarFactory {
public:
  virtual Car *buildWholeCar() = 0;

protected:
  virtual Tire *buildTire() = 0;
  virtual Body *buildBody() = 0;

private:
  Car *car;
};

class SimpleCarFactory : public CarFactory {
public:
  Car *buildWholeCar() {
    Car *car = new Car((string) "SimpleCar");
    car->setTire(buildTire());
    car->setBody(buildBody());
    return car;
  };

protected:
  Tire *buildTire() { return new SimpleTire(); };
  Body *buildBody() { return new SimpleBody(); };

private:
};

class LuxuryCarFactory : public CarFactory {
public:
  Car *buildWholeCar() {
    Car *car = new Car((string) "LuxuryCar");
    car->setTire(buildTire());
    car->setBody(buildBody());
    return car;
  };

protected:
  Tire *buildTire() { return new LuxuryTire(); };
  Body *buildBody() { return new LuxuryBody(); };
```

```
private:
};

} // namespace AFDP
```

## src/Car.cpp

```
#ifndef __io__
#define __io__
#include <iostream>
using std::cout;
using std::endl;
#endif // !__io__

#include <string>

using std::string;

namespace AFDP {

class Tire {
protected:
  string name;
  int pressure;

public:
  Tire(string n, int pressure) : name(n), pressure(pressure){};
  virtual string getName() { return name; }
  virtual int getPressure() { return pressure; }
};

class SimpleTire : public Tire {
public:
  SimpleTire() : Tire("SimpleTire", 75) {}
};

class LuxuryTire : public Tire {
public:
  LuxuryTire() : Tire("LuxuryTire", 100) {}
};

class Body {
protected:
```

```
  string name;
  int strength;

public:
  Body(string n, int strength) : name(n), strength(strength){};
  virtual string getName() { return name; }
  virtual int getStrength() { return strength; }
};

class SimpleBody : public Body {
public:
  SimpleBody() : Body("SimpleBody", 75) {}
};

class LuxuryBody : public Body {
public:
  LuxuryBody() : Body("LuxuryBody", 100) {}
};

class Car {
public:
  Car(string type) : name(type) {}

  void setTire(Tire *tire) {
    this->tire = tire;
    return;
  }
  void setBody(Body *body) {
    this->body = body;
    return;
  }

  void printDetails() {
    cout << endl << "Car: " << name << endl;
    cout << "Tire: " << tire->getName() << " Pressure: " << tire-
>getPressure()
         << endl;
    cout << "Body: " << body->getName() << " Strength: " << body-
>getStrength()
         << endl
         << endl;
    return;
  }

protected:
  string name;
```

```
    Tire *tire;
    Body *body;

private:
};

} // namespace AFDP
```