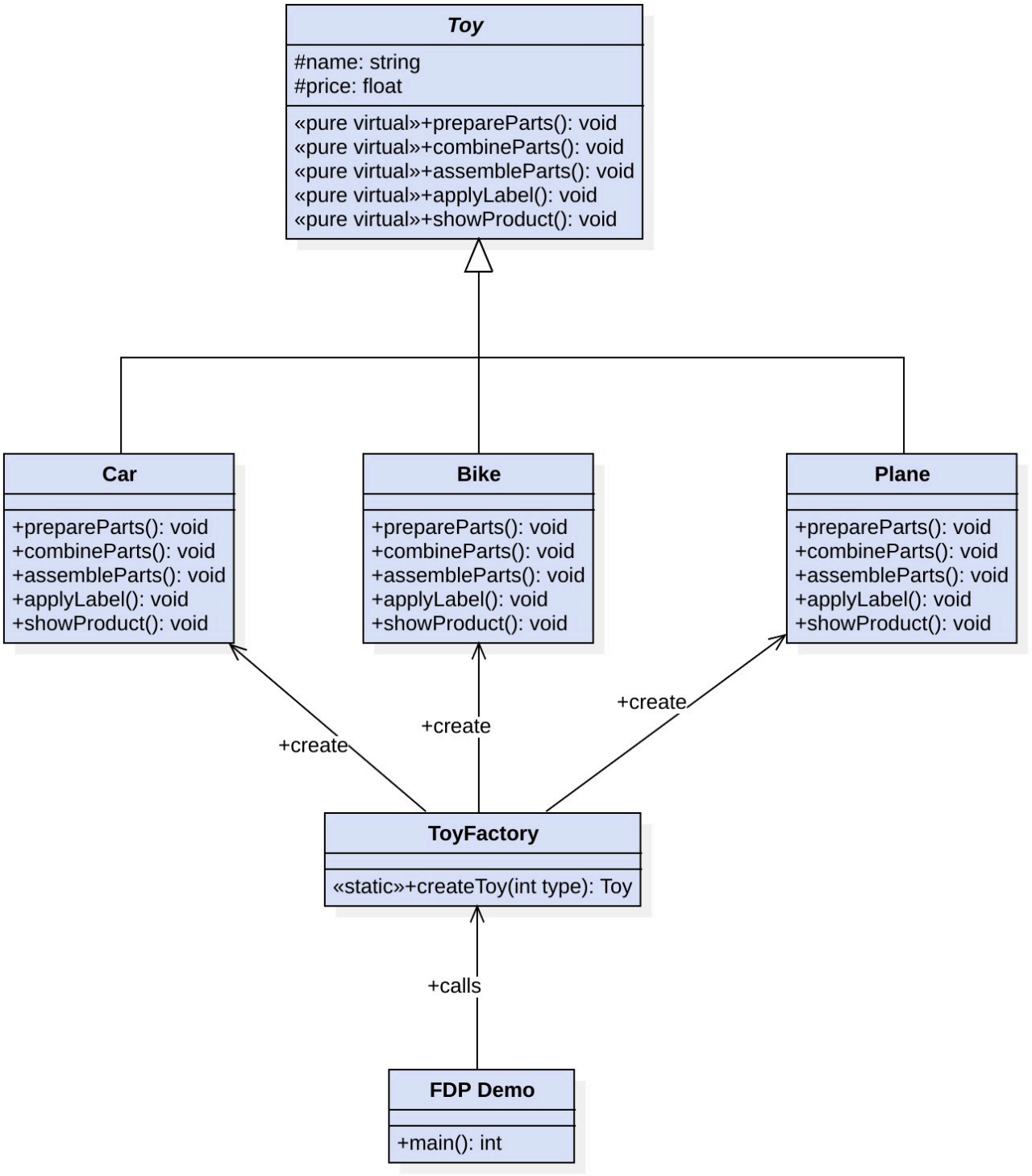
1. A factory creates objects for you rather than having to initialize the object directly.
2. 2 FDP - Factory Design Pattern is also known as a *Virtual Constructor*

C++ does not support Virtual Constructors. Instead the superclass constructor is used unless overwritten.

> A virtual call is a mechanism to get work done given partial information. In particular, "virtual" allows us to call a function knowing only an interfaces and not the exact type of the object. To create an object you need complete information. In particular, you need to know the exact type of what you want to create. Consequently, a "call to a constructor" cannot be virtual.
>
> - Bjarne Stroustrup, inventor of the C++ language

How to implement FDP

Define an interface or an abstract class for creating an object but let the subclasses decide which class has to initiate.



This allows us to create object dynamically at runtime instead of manually preparing them.

Advantages

- Fewer code changes to change the object creation process.
- Creating objects without exposing creation logic to the client.

- Benefiting from a common virtual constructor.

The fact that we don't expose object creation logic to the client makes this pattern well suited for libraries and such.

Reference Implementation

src/FDPDemo.cpp

```
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
#ifndef TOY
#define TOY
#include "../include/Toy.h"
#endif // !TOY
#include "../include/ToyFactory.h"
using FDP::Toy;
using FDP::ToyFactory;
int main(int argc, char *argv[]) {
  int type;
  while (true) {
    cout << endl << "Enter type number (1 ... 3) or Zero for exit" << endl;
    cin >> type;
    if (!type) {
      break;
    }
    Toy *v = ToyFactory::createToy(type);
    if (v) {
      v->showProduct();
      delete v;
    }
  }
  cout << "[ Exit ]" << endl;
  return 0;
}
```

include/Bike.h

```
#ifndef TOY
#define TOY
```

```
#include "Toy.h"
#endif
namespace FDP {
class Bike : public Toy {
public:
  void prepareParts();
  void combineParts();
  void assembleParts();
  void applyLabel();
  void showProduct();
private:
};
} // namespace FDP
```

src/Bike.cpp

```
#include "../include/Bike.h"
#include <iostream>
using std::cout;
using std::endl;
namespace FDP {
void Bike::prepareParts() {
  cout << "preparing bike parts" << endl;
  return;
}
void Bike::combineParts() {
  cout << "combining bike parts" << endl;
  return;
}
void Bike::assembleParts() {
  cout << "assembling bike parts" << endl;
  return;
}
void Bike::applyLabel() {
  cout << "applyig bike label" << endl;
  return;
}
void Bike::showProduct() {
  cout << "showing bike product" << endl;
  return;
}
} // namespace FDP
```

include/Car.h

```
#ifndef TOY
#define TOY
#include "Toy.h"
#endif // !TOY
namespace FDP {
class Car : public Toy {
public:
  Car();
  Car(string, float);
  void prepareParts();
  void combineParts();
  void assembleParts();
  void applyLabel();
  void showProduct();
  ~Car();
private:
};
} // namespace FDP
```

src/Car.cpp

```
#include "../include/Car.h"
#include <iostream>
using std::cout;
using std::endl;
namespace FDP {
Car::Car() {
  this->name = "default";
  this->price = 3.4f;
}
Car::Car(string name, float price) {
  this->name = name;
  this->price = price;
};
void Car::prepareParts() {
  cout << "preparing car '" << this->name << "' parts" << endl;
  return;
}
void Car::combineParts() {
  cout << "combining car parts" << endl;
  return;
```

```
}
void Car::assembleParts() {
  cout << "assembling car parts" << endl;
  return;
}
void Car::applyLabel() {
  cout << "applying car label" << endl;
  return;
}
void Car::showProduct() {
  cout << "showing car product" << endl;
  return;
}
Car::~Car() {}
} // namespace FDP
```

include/Plane.h

```
#ifndef TOY
#define TOY
#include "Toy.h"
#endif // !TOY
namespace FDP {
class Plane : public Toy {
public:
  void prepareParts();
  void combineParts();
  void assembleParts();
  void applyLabel();
  void showProduct();
private:
};
} // namespace FDP
```

src/Plane.cpp

```
#include "../include/Plane.h"
#include <iostream>
using std::cout;
using std::endl;
namespace FDP {
void Plane::prepareParts() {
```

```
  cout << "preparing plane '" << this->name << "' parts" << endl;
  return;
}
void Plane::combineParts() {
  cout << "combining plane parts" << endl;
  return;
}
void Plane::assembleParts() {
  cout << "assembling plane parts" << endl;
  return;
}
void Plane::applyLabel() {
  cout << "applying plane label" << endl;
  return;
}
void Plane::showProduct() {
  cout << "showing plane product" << endl;
  return;
}
} // namespace FDP
```

include/Toy.h

```
#include <string>
using std::string;
namespace FDP {
class Toy {
public:
  Toy();
  Toy(string, float);
  virtual void prepareParts();
  virtual void combineParts();
  virtual void assembleParts();
  virtual void applyLabel();
  virtual void showProduct();
  virtual ~Toy();
private:
protected:
  string name;
  float price;
};
} // namespace FDP
```

src/Toy.cpp

```
#include "../include/Toy.h"
namespace FDP {
Toy::Toy() {
  this->name = "default";
  this->price = 3.4f;
}
Toy::Toy(string name, float price) {
  this->name = name;
  this->price = price;
};
void Toy::prepareParts() {}
void Toy::combineParts() {}
void Toy::assembleParts() {}
void Toy::applyLabel() {}
void Toy::showProduct() {}
Toy::~Toy() {}
} // namespace FDP
```

include/ToyFactory.h

```
#include "Bike.h"
#include "Car.h"
#include "Plane.h"
#ifndef TOY
#define TOY
#include "Toy.h"
#endif // !TOY
namespace FDP {
class ToyFactory {
public:
  static Toy *createToy(int type);
private:
};
} // namespace FDP
```

src/ToyFactory.cpp

```
#include "../include/ToyFactory.h"
#include <cstddef>
```

```
#include <iostream>
using std::cout;
using std::endl;
namespace FDP {
Toy *ToyFactory::createToy(int type) {
  Toy *toy = NULL;
  switch (type) {
  case 1: {
    toy = new Car("fancy car", 2.3f);
    break;
  }
  case 2: {
    toy = new Bike;
    break;
  }
  case 3: {
    toy = new Plane;
    break;
  }
  default: {
    cout << "invalid toy type. Please enter valid type" << endl;
    return NULL;
  }
  }
  toy->prepareParts();
  toy->combineParts();
  toy->assembleParts();
  toy->applyLabel();
  return toy;
}
} // namespace FDP
```