Prototype 패턴

타입

Creational Pattern

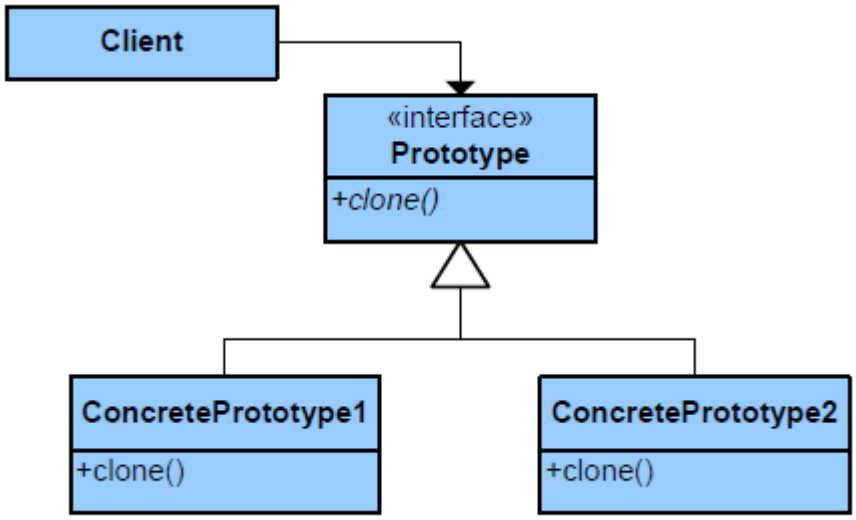
문제

새로운 객체를 만들려고할 때 드는 시간과 비용이 크다.

해결

clone() 가상 함수를 가지는 추상 클래스를 정의하고 파생 클래스에 clone() 함수를 통해 자기 자신을 복사하게 한다.

클래스 다이어그램



예제

prototype1.cpp

```
/**
 * Implementation of Prototype Method
 **/
#include <iostream>
#include <map>
```

```
#include <string>

using namespace std;

enum RECORD_TYPE_en
{
  CAR,
  BIKE,
  PERSON
};

/**
 * Record is the base Prototype
 */

class Record
{
  public :

    Record() {}

    virtual ~Record() {}

    virtual Record* clone()=0;

    virtual void print()=0;
};

/**
 * CarRecord is a Concrete Prototype
 */

class CarRecord : public Record
{
  private:
    string m_carName;
    int m_ID;

  public:
    CarRecord(string carName, int ID)
      : Record()
      , m_carName(carName)
      ,m_ID(ID)
    {
    }

    CarRecord(const CarRecord& carRecord)
      : Record(carRecord)//call the base default copy  constructor
    {
      m_carName = carRecord.m_carName;
```

```
      m_ID = carRecord.m_ID;
    }

    ~CarRecord() {}

    Record* clone()
    {
      return new CarRecord(*this);
    }

    void print()
    {
      cout << "Car Record" << endl
        << "Name  : " << m_carName << endl
        << "Number: " << m_ID << endl << endl;
    }
};


/**
 * BikeRecord is the Concrete Prototype
 */

class BikeRecord : public Record
{
  private :
    string m_bikeName;

    int m_ID;

  public :
    BikeRecord(string bikeName, int ID)
      : Record()
      , m_bikeName(bikeName)
      , m_ID(ID)
    {
    }

    BikeRecord(const BikeRecord& bikeRecord)
      : Record(bikeRecord)
    {
      m_bikeName = bikeRecord.m_bikeName;
      m_ID = bikeRecord.m_ID;
    }

    ~BikeRecord() {}

    Record* clone()
    {
      return new BikeRecord(*this);
    }
```

```
    void print()
    {
      cout << "Bike Record" << endl
        << "Name  : " << m_bikeName << endl
        << "Number: " << m_ID << endl << endl;
    }
};


/**
 * PersonRecord is the Concrete Prototype
 */

class PersonRecord : public Record
{
  private :
    string m_personName;

    int m_age;

  public :
    PersonRecord(string personName, int age)
      : Record()
      , m_personName(personName)
      , m_age(age)
    {
    }

    PersonRecord(const PersonRecord& personRecord)
      : Record(personRecord)
    {
      m_personName = personRecord.m_personName;
      m_age = personRecord.m_age;
    }

    ~PersonRecord() {}

    Record* clone()
    {
      return new PersonRecord(*this);
    }

  void print()
  {
    cout << "Person Record" << endl
      << "Name : " << m_personName << endl
      << "Age  : " << m_age << endl << endl ;
  }
};
```

```
/**
 * RecordFactory is the client
 */

class RecordFactory
{
  private :
    map<RECORD_TYPE_en, Record* > m_recordReference;

  public :
    RecordFactory()
    {
      m_recordReference[CAR]  = new CarRecord("Ferrari", 5050);
      m_recordReference[BIKE] = new BikeRecord("Yamaha", 2525);
      m_recordReference[PERSON] = new PersonRecord("Tom", 25);
    }

    ~RecordFactory()
    {
      delete m_recordReference[CAR];
      delete m_recordReference[BIKE];
      delete m_recordReference[PERSON];
    }

    Record* createRecord(RECORD_TYPE_en enType)
    {
      return m_recordReference[enType]->clone();
    }
};

int main()
{
  RecordFactory* poRecordFactory = new RecordFactory();

  Record* poRecord;
  poRecord = poRecordFactory->createRecord(CAR);
  poRecord->print();
  delete poRecord;

  poRecord = poRecordFactory->createRecord(BIKE);
  poRecord->print();
  delete poRecord;

  poRecord = poRecordFactory->createRecord(PERSON);
  poRecord->print();
  delete poRecord;

  delete poRecordFactory;
  return 0;
```

```
}
```

prototype2.cpp

```
class CPrototypeMonster
 {
 protected:
     CString           _name;
 public:
     CPrototypeMonster();
     CPrototypeMonster( const CPrototypeMonster& copy );
     virtual ~CPrototypeMonster();

     virtual CPrototypeMonster*    Clone() const=0; // This forces
every derived class to provide an overload for this function.
     void         Name( CString name );
     CString    Name() const;
 };

 class CGreenMonster : public CPrototypeMonster
 {
 protected:
     int          _numberOfArms;
    double       _slimeAvailable;
 public:
     CGreenMonster();
     CGreenMonster( const CGreenMonster& copy );
     ~CGreenMonster();

     virtual CPrototypeMonster*    Clone() const;
     void  NumberOfArms( int numberOfArms );
     void  SlimeAvailable( double slimeAvailable );

     int          NumberOfArms() const;
     double       SlimeAvailable() const;
 };

 class CPurpleMonster : public CPrototypeMonster
 {
 protected:
     int          _intensityOfBadBreath;
     double       _lengthOfWhiplikeAntenna;
 public:
     CPurpleMonster();
     CPurpleMonster( const CPurpleMonster& copy );
     ~CPurpleMonster();

     virtual CPrototypeMonster*    Clone() const;
```

```
    void  IntensityOfBadBreath( int intensityOfBadBreath );
    void  LengthOfWhiplikeAntenna( double lengthOfWhiplikeAntenna );

    int       IntensityOfBadBreath() const;
    double    LengthOfWhiplikeAntenna() const;
};

class CBellyMonster : public CPrototypeMonster
{
protected:
    double      _roomAvailableInBelly;
public:
    CBellyMonster();
    CBellyMonster( const CBellyMonster& copy );
    ~CBellyMonster();

    virtual CPrototypeMonster*    Clone() const;

    void       RoomAvailableInBelly( double roomAvailableInBelly );
    double     RoomAvailableInBelly() const;
};

CPrototypeMonster* CGreenMonster::Clone() const
{
    return new CGreenMonster(*this);
}

CPrototypeMonster* CPurpleMonster::Clone() const
{
    return new CPurpleMonster(*this);
}

CPrototypeMonster* CBellyMonster::Clone() const
{
    return new CBellyMonster(*this);
}

// Client

void DoSomeStuffWithAMonster( const CPrototypeMonster* originalMonster
)
{
    CPrototypeMonster* newMonster = originalMonster->Clone();
    ASSERT( newMonster );

    newMonster->Name("MyOwnMonster");
    // Add code doing all sorts of cool stuff with the monster.
    delete newMonster;
}
```

참고

http://en.wikibooks.org/wiki/C%2B%2B_Programming/Code/Design_Patterns#Prototype