Языки программирования

Лекция 1. **Позднее и раннее связывание. Полиморфизм. Основные понятия.**

Понятие позднего и раннего связывания

При изучении темы полиморфизма важно понять понятие позднего и раннего связывания, которое используется компилятором при построении кода программы в случае наследования.

Если классы образовывают иерархию наследования, то при обращении к элементам класса, компилятор может реализовывать один из двух возможных способов связывания кода:

- ***Раннее связывание*** – связанное с формированием кода на этапе компиляции. При раннем связывании, программный код формируется на основе известной информации о типе (класс) ссылки. Как правило, это ссылка на базовый класс в иерархии классов.
- ***Позднее связывание*** – связанное с формированием кода на этапе выполнения. Если в иерархии классов встречается цепочка виртуальных методов (с помощью слов virtual, override), то компилятор строит так называемое позднее связывание. При позднем связывании вызов метода происходит на основании типа объекта, а не типа ссылки на базовый класс. Позднее связывание используется, если нужно реализовать *полиморфизм*

Понятие позднего и раннего связывания

Выбор того или иного вида связывания для каждого отдельного элемента (метода, свойства, индексатора и т.п.) определяется компилятором по следующим правилам:

- если в иерархии унаследованных классов объявляется невиртуальный элемент, то реализуется раннее связывание;
- если в иерархии унаследованных классов объявляется виртуальный элемент, то выполняется позднее связывание. Виртуальный элемент в базовом классе обозначается ключевым словом virtual, во всех унаследованных классах ключевым словом override.

Необходимые условия для реализации позднего связывания:

- классы должны образовывать иерархию наследования;
- в классах должны быть методы с одинаковой сигнатурой. Элементы (методы) производных классов должны перекрывать (override) соответствующие элементы (методы) базовых классов;
- элементы (методы) класса должны быть виртуальными, то есть должны быть обозначены ключевыми словами virtual, override.

Раннее связывание

В случае раннего связывания, как только компилятор встречает строку

```
A ref;
```

происходит объявление ссылки ref, которая имеет тип базового класса A. Дальнейшее присваивание

```
refA = objB;
```

связывает ссылку с объектом objB, однако тип ссылки устанавливается A. Поэтому вызов

```
ref.Print();
```

вызовет метод Print() класса A.

Позднее связывание

В случае позднего связывания, сначала на основе описания классов A, B компилятор определяет, что метод Print() есть виртуальным. Для виртуального метода компилятор строит таблицу виртуальных методов Print(), которая содержит смещение адресов каждого виртуального метода для каждого класса иерархии.

После строки

```
A ref;
```

формируется связывание ссылки ref с типом A. После присваивания

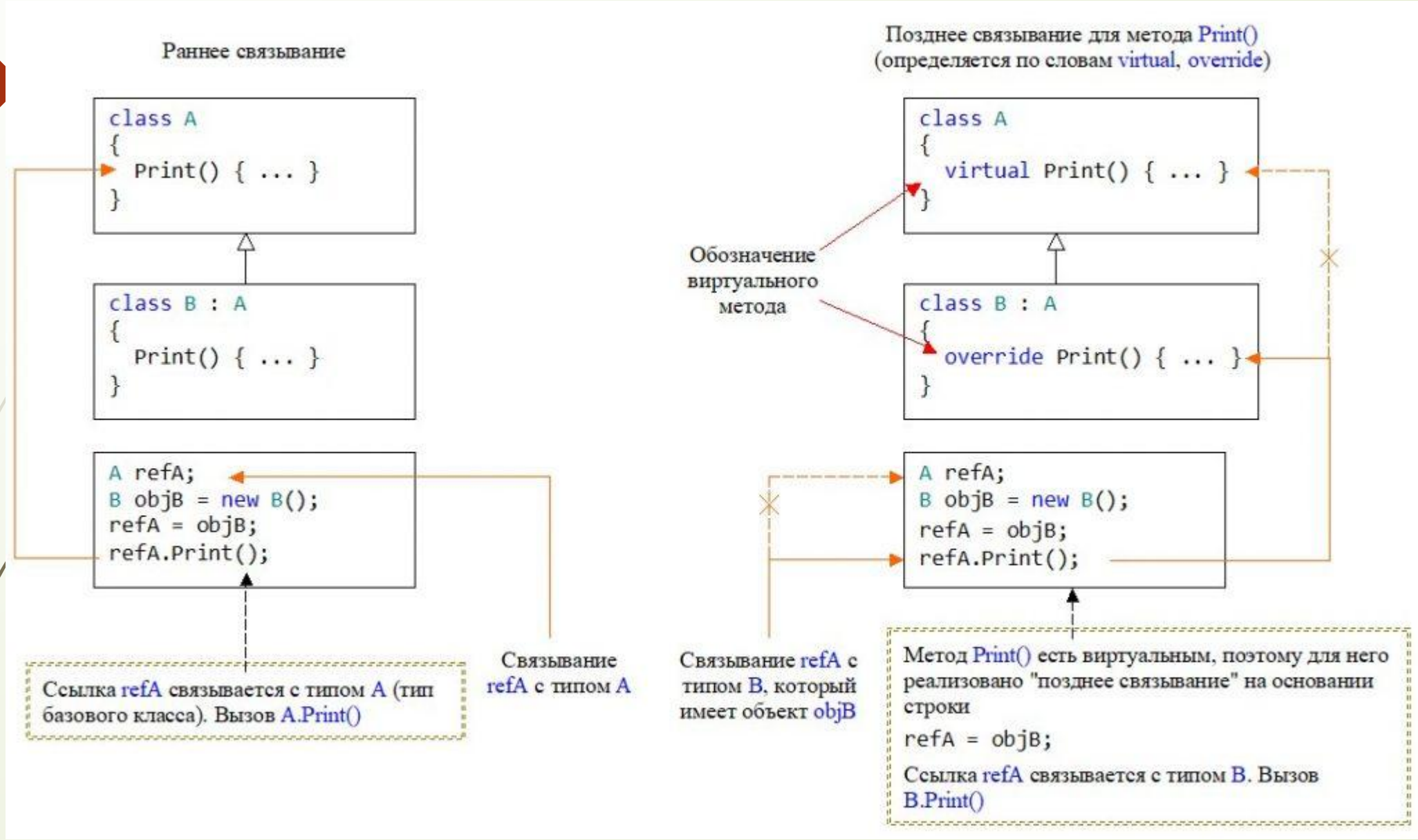
```
ref = objB;
```

компилятор присваивает ссылке ref адрес экземпляра objB и определяет тип связывания как тип B (поскольку метод Print() виртуальный). За основу берется тип объекта. В результате ссылка ref связывается с методом Print(), который реализован в классе B (а не в классе A) – выполняется так называемое «позднее связывание».

Как следствие, после вызова

```
ref.Print();
```

будет вызван метод Print() класса B.


Раннее связывание
class A
{
Print() { ... }
}
class B : A
{
Print() { ... }
}
A refA;
B objB = new B();
refA = objB;
refA.Print();
Ссылка refA связывается с типом A (тип базового класса). Вызов A.Print()
Связывание refA с типом A
Позднее связывание для метода Print()
(определяется по словам virtual, override)
class A
{
virtual Print() { ... }
}
Обозначение виртуального метода
class B : A
{
override Print() { ... }
}
A refA;
B objB = new B();
refA = objB;
refA.Print();
Связывание refA с типом B, который имеет объект objB
Метод Print() есть виртуальным, поэтому для него реализовано "позднее связывание" на основании строки
refA = objB;
Ссылка refA связывается с типом B. Вызов B.Print()

Что такое полиморфизм? Динамический полиморфизм

Полиморфизм – это свойство программного кода изменяться в зависимости от ситуации, которая возникает в момент выполнения программы.

Главный принцип полиморфизма – один интерфейс, много реализаций (методов). В терминах языка программирования, *полиморфизм* – это возможность с помощью ссылки на базовый класс обращаться к элементам (методов) экземпляров унаследованных классов единым унифицированным способом.

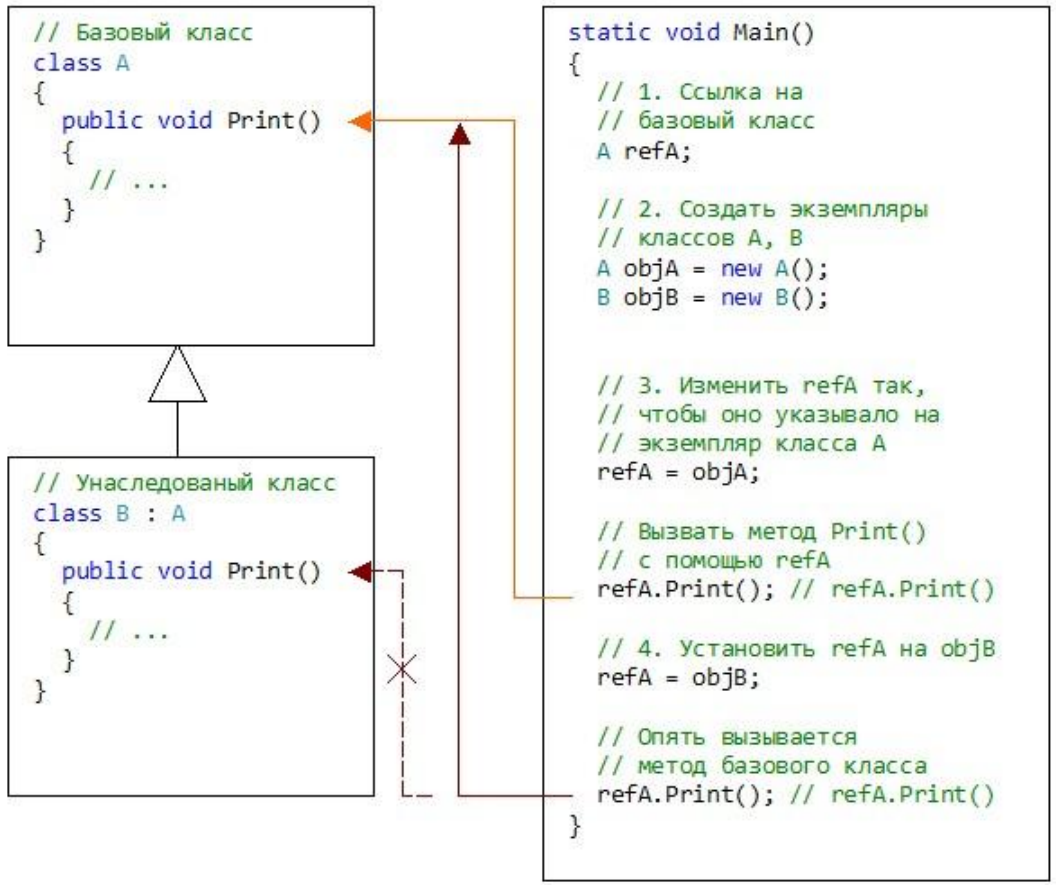
Использование преимуществ полиморфизма возможно в ситуациях:

- когда классы образовывают иерархию с помощью концепции наследования;
- когда в классах, которые образовывают иерархию, есть элементы (методы, свойства и т.п.) с одинаковой сигнатурой. В таких случаях возникает понятие «переопределение метода» (method override).

Полиморфизм. Ключевые слова

- В языке программирования C# полиморфизм обеспечивается с помощью ключевых слов virtual и override. Благодаря использованию этих ключевых слов обеспечивается *динамический полиморфизм*. Термин «динамический» означает, что вызов виртуального элемента осуществляется динамично во время выполнения программы в зависимости от типа объекта, для которого этот элемент вызывается.

1. Полиморфизм не поддерживается для метода Print()

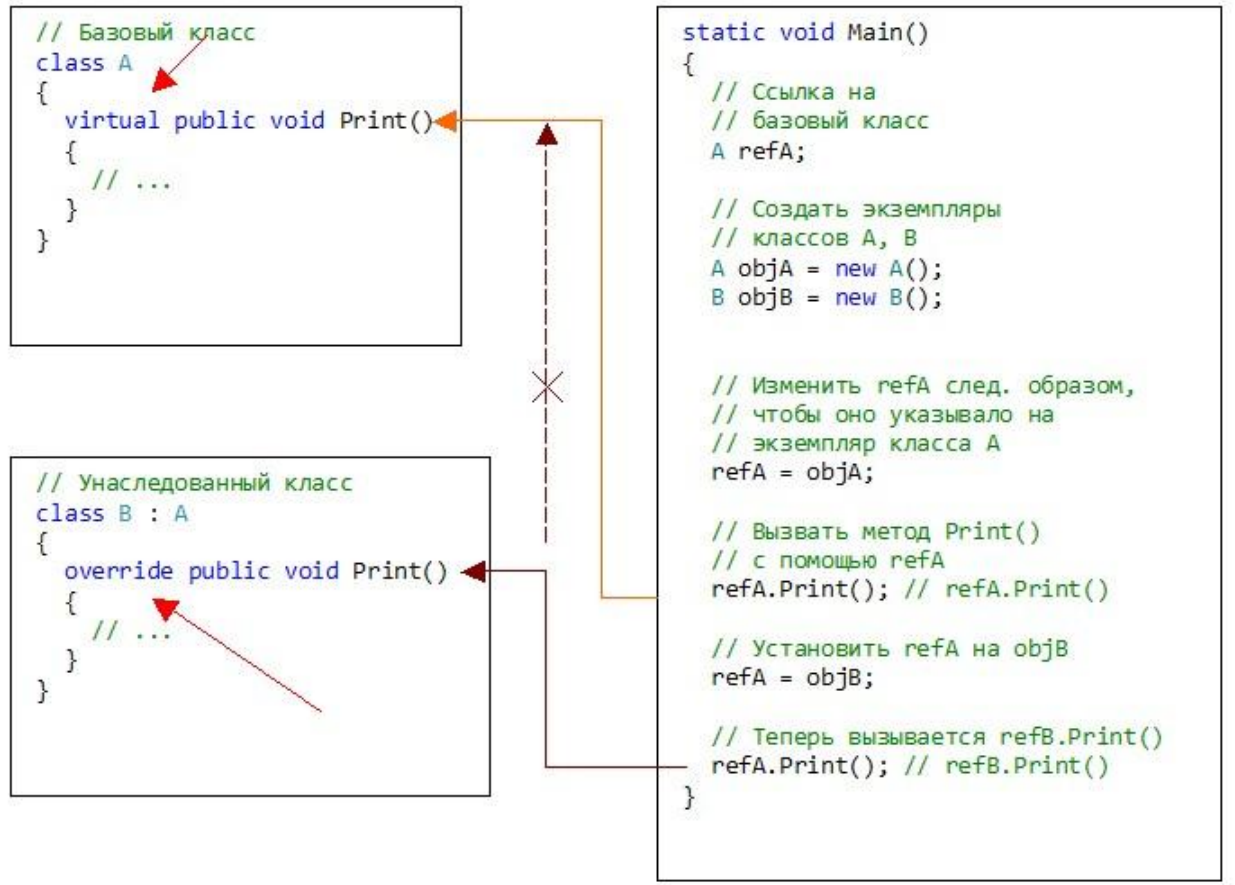


```
// Базовый класс
class A
{
  public void Print()
  {
    // ...
  }
}
```

```
// Унаследованый класс
class B : A
{
  public void Print()
  {
    // ...
  }
}
```

```
static void Main()
{
  // 1. Ссылка на
  // базовый класс
  A refA;

  // 2. Создать экземпляры
  // классов A, B
  A objA = new A();
  B objB = new B();


  // 3. Изменить refA так,
  // чтобы оно указывало на
  // экземпляр класса A
  refA = objA;

  // Вызвать метод Print()
  // с помощью refA
  refA.Print(); // refA.Print()

  // 4. Установить refA на objB
  refA = objB;

  // Опять вызывается
  // метод базового класса
  refA.Print(); // refA.Print()
}
```

2. Полиморфизм поддерживается для метода Print() (ключевые слова virtual, override)



```
// Базовый класс
class A
{
  virtual public void Print()
  {
    // ...
  }
}
```

```
// Унаследованный класс
class B : A
{
  override public void Print()
  {
    // ...
  }
}
```

```
static void Main()
{
  // Ссылка на
  // базовый класс
  A refA;

  // Создать экземпляры
  // классов A, B
  A objA = new A();
  B objB = new B();


  // Изменить refA след. образом,
  // чтобы оно указывало на
  // экземпляр класса A
  refA = objA;

  // Вызвать метод Print()
  // с помощью refA
  refA.Print(); // refA.Print()

  // Установить refA на objB
  refA = objB;

  // Теперь вызывается refB.Print()
  refA.Print(); // refB.Print()
}
```

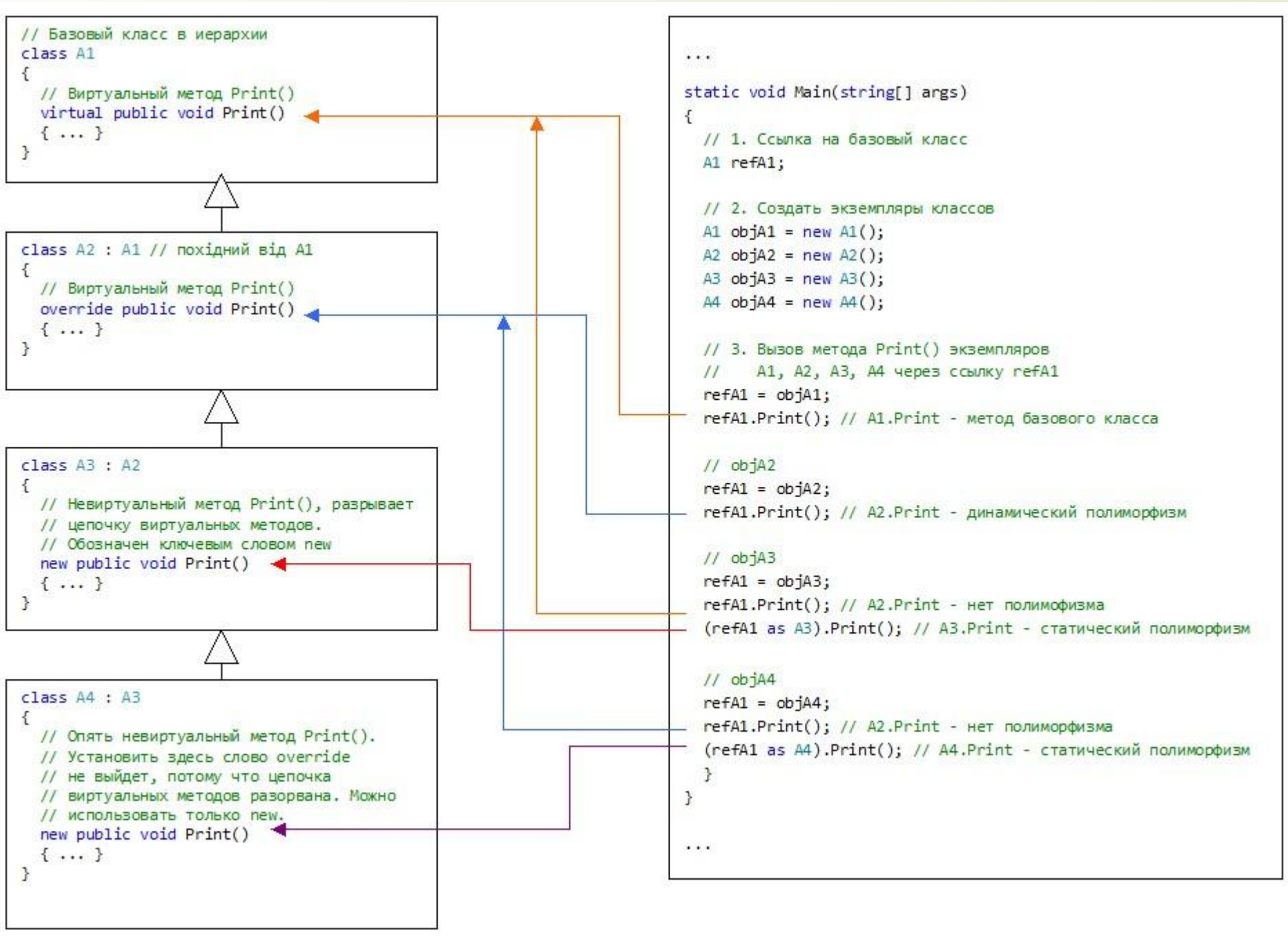
Какие требования накладываются на элемент класса для того, чтобы он поддерживал полиморфизм?

Для того, чтобы элемент класса (например метод) поддерживал полиморфизм, его нужно сделать виртуальным. Чтобы элемент класса был виртуальным, нужно выполнить следующие требования:

- в базовом классе этот элемент (метод, свойство) должен быть обозначен как virtual или abstract. Ключевое слово abstract также делает элемент виртуальным. Это слово используется, если элемент класса есть абстрактным.
- в производных классах одноименные элементы должны быть обозначены как override. Если в производном классе нужно реализовать невиртуальный метод, имя которого совпадает с виртуальным методом базового класса, то этот метод обозначается ключевым словом new

Использование ключевого слова new в цепочке виртуальных методов.

- Как известно, элемент класса, который объявлен виртуальным (virtual), передает возможность реализовать полиморфизм в одноименных элементах унаследованных классов. Таким образом, виртуальные элементы образовывают цепочку вниз по иерархии.
- Для того, чтобы элемент класса, который переопределяет (override) виртуальный элемент базового класса, не поддерживал полиморфизм нужно указать ключевое слово new. Если в цепочке одноименных виртуальных методов встречается один невиртуальный метод (с ключевым словом new) то этот метод разрывает цепочку.


// Базовый класс в иерархии
class A1
{
// Виртуальный метод Print()
virtual public void Print()
{ ... }
}
class A2 : A1 // похідний від A1
{
// Виртуальный метод Print()
override public void Print()
{ ... }
}
class A3 : A2
{
// Невиртуальный метод Print(), разрывает
// цепочку виртуальных методов.
// Обозначен ключевым словом new
new public void Print()
{ ... }
}
class A4 : A3
{
// Опять невиртуальный метод Print().
// Установить здесь слово override
// не выйдет, потому что цепочка
// виртуальных методов разорвана. Можно
// использовать только new.
new public void Print()
{ ... }
}
...
static void Main(string[] args)
{
// 1. Ссылка на базовый класс
A1 refA1;
// 2. Создать экземпляры классов
A1 objA1 = new A1();
A2 objA2 = new A2();
A3 objA3 = new A3();
A4 objA4 = new A4();
// 3. Вызов метода Print() экземпляров
// A1, A2, A3, A4 через ссылку refA1
refA1 = objA1;
refA1.Print(); // A1.Print - метод базового класса
// objA2
refA1 = objA2;
refA1.Print(); // A2.Print - динамический полиморфизм
// objA3
refA1 = objA3;
refA1.Print(); // A2.Print - нет полимофизма
(refA1 as A3).Print(); // A3.Print - статический полиморфизм
// objA4
refA1 = objA4;
refA1.Print(); // A2.Print - нет полиморфизма
(refA1 as A4).Print(); // A4.Print - статический полиморфизм
}
}
...