# Invocation virtual property from constructor (CSharp)

|  |
| --- |
| using System;  internal class A  {  public virtual String ClassName { get { return "A"; }}  public A()  {  Console.WriteLine($"A:{ClassName}");  }  }  internal class B : A  {  public override String ClassName { get { return "B"; }}  public B()  {  Console.WriteLine($"B:{ClassName}");  }  }  internal class Program  {  private static void Main(string[] args)  {  Console.WriteLine($"Main:{(new A()).ClassName}");  Console.WriteLine($"Main:{(new B()).ClassName}");  }  } |
| A:A Main:A A:**B** B:B Main:B |

# Invocation virtual method from constructor (C++)

|  |
| --- |
| #include <iostream>  #include <string>  using namespace std;  class A  {  public:  A()  {  cout << "A:" << ClassName() << endl;  }  public:  virtual string ClassName()  {  return "A";  }  };  class B: public A  {  public:  B():  A()  {  cout << "B:" << ClassName() << endl;  }  public:  string ClassName() override  {  return "B";  }  };  int main()  {  cout << "Main:" << (new A())->ClassName() << endl;  cout << "Main:" << (new B())->ClassName() << endl;  } |
| A:A Main:A A:**A** B:B Main:B |

# Multiple inheritance (C++)

|  |
| --- |
| #include <iostream>  #include <string>  using namespace std;  class A  {  public:  A()  {  }  };  class B: public A  {  public:  B():  A()  {  }  };  class C: public A  {  public:  C():  A()  {  }  };  class D:  public B,  public C  {  };  int main()  {  auto d = new D();  printf("D:%p\n",d);  printf("B:%p\n",static\_cast<B\*>(d));  printf("C:%p\n",static\_cast<C\*>(d));  printf("A-B:%p\n",static\_cast<A\*>(static\_cast<B\*>(d)));  printf("A-C:%p\n",static\_cast<A\*>(static\_cast<C\*>(d)));  } |
| D:0000021830191910  B:0000021830191910  C:0000021830191911  A-B:**0000021830191910**  A-C:**0000021830191911** |

# Virtual inheritance (C++)

|  |
| --- |
|  |