Тимофеева Наталья

М8О-202Б-19

ООП C#

Лабораторная работа № 3

**Описание работы**

1. Изучите категорию наследование и принцип подстановки. Какие разновидности этой категории есть. Для каких целей используется принцип подстановки? Что он дает?

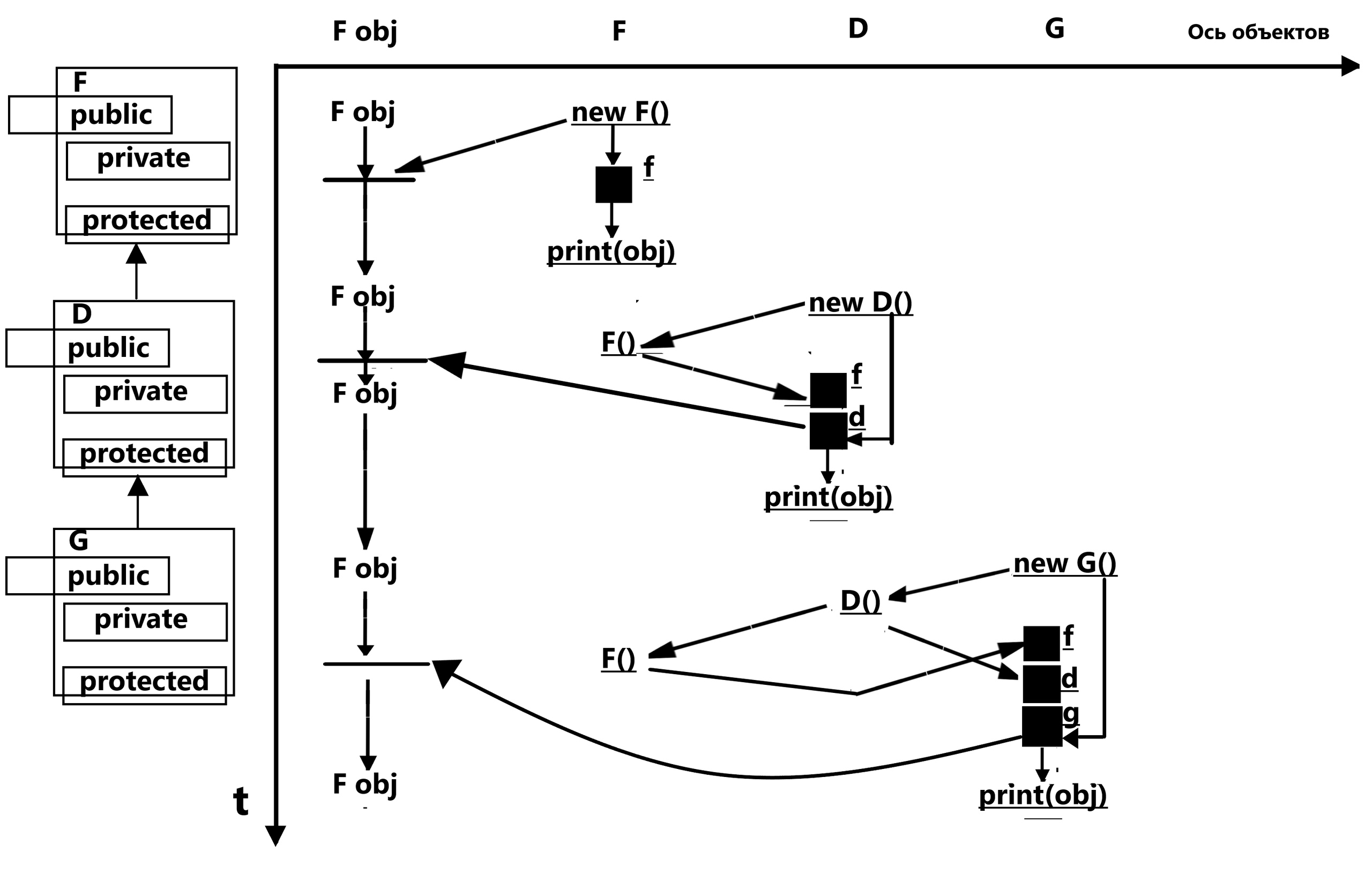
2. Дайте определение замещаемой функции. Как создаются и разрушаются объекты. Объясните утечку памяти. Как определить тип объекта по полиморфному указателю.

3. Реализуйте принцип подстановки в отдельном проекте для иерархии F-D-G, где F – суперкласс, а каждый класс имеет некоторое свойство protected и замещаемую функцию, которая возвращает его значение. Свойство инициализируется в конструкторе. Дайте определение модификатору правила видимости protected. Проиллюстрируйте свои объяснения к вопросам пункта 2. на реализованном примере.

Что измениться, если убрать или добавить модификатор virtual: для диструктора суперкласса, для функции суперкласса.

4. Оформите работу. На графе укажите вид категорий. Напишите ответы на вопросы к пункту 1,2 и 3. Сохраните результаты лабораторной.

**Граф**



**Программа**

using System;

namespace Lab3 {

class F {

protected int f = 80;

int numf = 0;

public F() {

Console.WriteLine("F def constr {0}", GetHashCode());

}

public F(int f, int numf) {

this.f = f;

this.numf = numf;

Console.WriteLine("F parm constr {0}, f = {1}, numf = {2}", GetHashCode(), f, numf);

}

public F(F of) {

f = of.f;

numf = of.numf;

Console.WriteLine("F copy constr {0} from numf {1}", GetHashCode(), of.GetHashCode());

}

public virtual void print() {

Console.WriteLine("Method of F, f = {0}", f);

}

}

class D : F{

protected int d = 31;

int numd = 1;

public D() {

Console.WriteLine("D def constr {0}", GetHashCode());

}

public D(int d, int numd, int f, int numf):base(f, numf) {

this.d = d;

this.numd = numd;

Console.WriteLine("D parm constr {0}, d = {1}, numd = {2}", GetHashCode(), d, numd);

}

public D(D od):base(od) {

d = od.d;

numd = od.numd;

Console.WriteLine("D copy constr {0} from numf {1}", GetHashCode(), od.GetHashCode());

}

public override void print() {

Console.WriteLine("Method of D, d = {0}, f = {1}", d, f);

}

}

class G : D {

int g = 67;

public G() {

Console.WriteLine("G def constr {0}", GetHashCode());

}

public G(int g, int d, int numd, int f, int numf):base(d, numd, f, numf) {

this.g = g;

Console.WriteLine("G parm constr {0}, g = {1}", GetHashCode(), g);

}

public G(G og):base(og) {

g = og.g;

Console.WriteLine("G copy constr {0} from numf {1}", GetHashCode(), og.GetHashCode());

}

public override void print() {

Console.WriteLine("Method of G, g = {0}, d = {1}, f = {2}", g, d, f);

}

}

class Program {

static void Main()

{

F objF1 = new F();

Console.WriteLine();

F objF2 = new F(35, 77);

Console.WriteLine();

F objF3 = new F(objF1);

Console.WriteLine();

D objD1 = new D();

Console.WriteLine();

D objD2 = new D(85, 41, 7, 39);

Console.WriteLine();

D objD3 = new D(objD1);

Console.WriteLine();

G objG1 = new G();

Console.WriteLine();

G objG2 = new G(10, 22, 93, 17, 55);

Console.WriteLine();

G objG3 = new G(objG1);

Console.WriteLine();

objF3.print();

objD3.print();

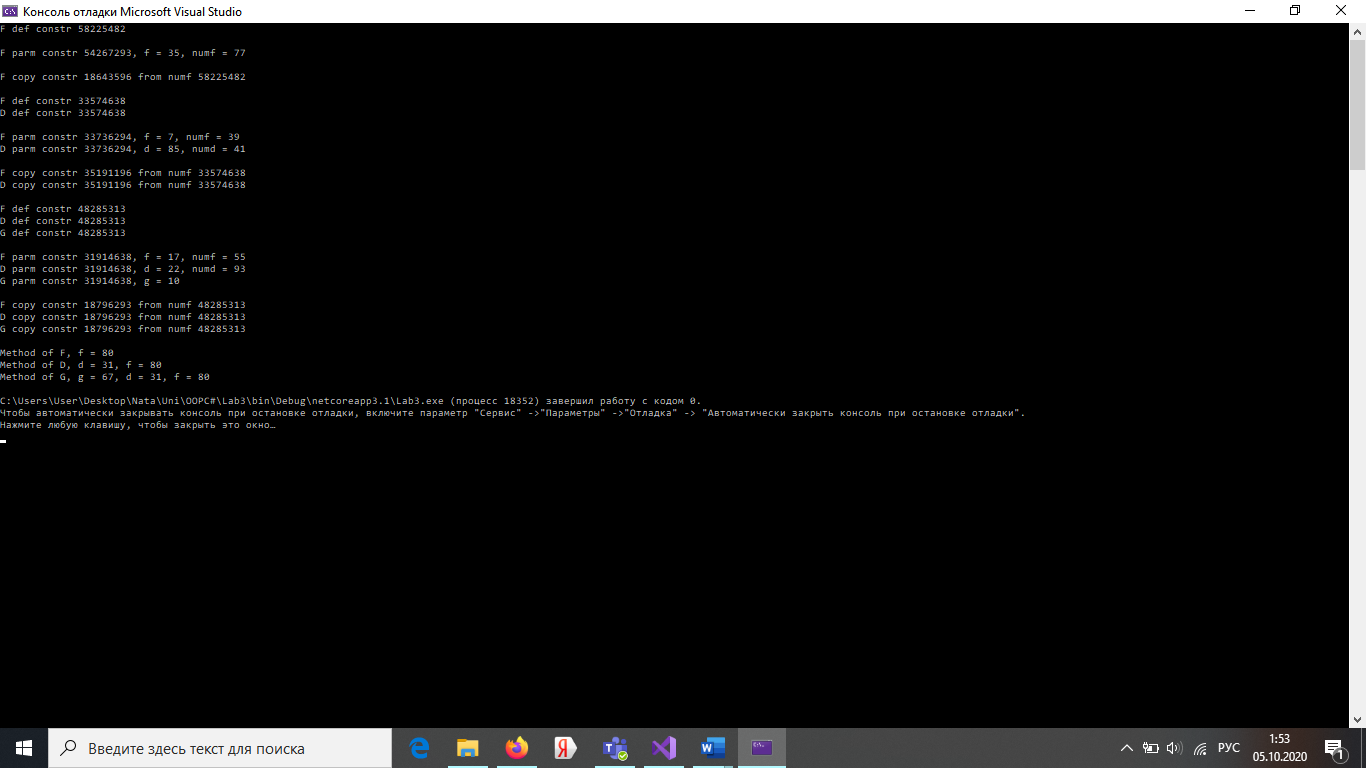
objG3.print();

}

}

}

**Вывод с консоли**



**Вывод**

Наследники и базовый класс в памяти являются одним и тем же объектом. Наследный класс копирует данные и функциональность класса-родителя.

Виртуальная функция отличается об обычной функции тем, что для обычной функции связывание вызова функции с ее определением осуществляется на этапе компиляции. Для виртуальных функций это происходит во время выполнения программы.