Позивање конструктора основне класе

Поред метода које наслеђује, изведена класа аутоматски обухвата сва поља из основне класе (надкласе).
За ова поља је обично потребна иницијализација када се прави објекат.
Ову врсту иницијализације обично се изводи у конструктору.
Најбоље је да конструктор у изведеној класи позива конструктор основне класе као део иницијализације, чиме се омогућава да конструктор основне класе обави све додатне иницијализације које су му потребне.
Може се навести кључна реч base за позивање конструктора основне класе када дефинишете конструктор за потклasu:

```
class Sisari
{
	public Sisari(string ime)
	{
		//...
	}
	//...
}
class Konj: Sisari
{
	public Konj(string ime) : base(ime)
	{
		//...
	}
	//...
}
```

Ако се експлицитно не позове конструктор основне класе у конструктору изведене класе, компајлер покушава да неприметно уметне позив на подразумевани конструктор основне класе пре него што изврши код у конструктору изведене класе:

```
class Konj: Sisari
{
	public Konj(string ime)
	{
		//...
	}
	//...
}
```

преправља у:

```
class Konj: Sisari
{
	public Konj(string ime): base()
	{
		//...
```

```
        }
        //...
}
```

Ово ради ако класа Sisari има јавни подразумевани конструктор.
Међутим, немају све класе јавни подразумевани конструктор и у том случају, ако се заборави да се позове исправан конструктор основне класе, као последица појавиће се грешка за време компајлирања.

Додељивање класа

У досадашњим примерима се видело како се декларише промењива коришћењем типа класе и како се користи службена реч new за прављење објеката.
Такође, није могуће објекат једног типа доделити промењивој која је декларисана као другачији тип података.
Следећи код није допуштен:

```
using System;
namespace ProjekatCS002
{
    class Sisari
    {
        //...
    }
    class Konj: Sisari
    {
        //...
    }
    class Kit: Sisari
    {
        //...
    }
    class Program
    {
        public static void Main()
        {
            Konj mojKonj = new Konj (...);
            Kit mojKit = mojKonj; //greska - razliciti tipovi
        }
    }
}
```

Међутим, могуће је указати на неки објекат из промењиве другачијег типа све док је тип који је користи класа на вишем нивоу у хијерархији наслеђивања; следећи исказ је дозвољен:

```
Konj mojKonj = new Konj (...);
Sisari mojSisar = mojKonj; //dozvoljeno, Sisari je osnovna klasa klasi
                           Konj
```

Према логици, сви Konji су Sisari, тако да безбедно можемо да доделимо објекат типа Konj промењивој типа Sisari.

Хијерархија наслеђивања значи да класу Konj можемо да замислимо као посебан тип класе Sisari – она има све што има Sisari, уз неколико додатних делова дефинисаних методама и пољима.
Такође, можемо да направимо да промењива типа Sisari указује на објекат Kit.
Међутим, овде постоји једно значајно ограничење: када се указује на објекат Konj или Kit коришћењем промењиве Sisari, можете да приступате само методама и пољима које су дефинисане класом Sisari.
Ниједна од додатних метода дефинисаних класом Konj или Kit није видљива преко класе Sisari:

```
Konj mojKonj = new Konj (...);
Sisari mojSisar = mojKonj;
mojSisar.Disanje(); //u redu - Disanje je deo klase Sisari
mojSisar.Kasanje(); //nije u redu - Kasanje nije deo klase Sisari
```

Ово објашњење описује зашто се скоро све може доделити промењивој типа object (object је псеудоним за System.Object, све класе наслеђују System.Object директно или индиректно).
Обрнута ситуација није тачна, не може се без задршке да објекат Sisar доделите промењивој Konj:

```
Sisari mojSisar = new Sisari (...);
Konj mojKonj = mojSisar; //greska
```

Ово изгледа као чудно ограничење, али сви објекти Sisari нису Konj – неки могу да буду Kit.
Објекат Sisari можете да доделите промењивој Konj ако пре тога проверите да ли је одређени Sisar стварно Konj, коришћењем оператора is и as или коришћењем претварања типова.
Следећи пример користи оператор as како би проверио да ли mojSisar указује на Konj.
Ако је тако, додељивање на mojKonjPonovo има за последицу то да mojKonjPonovo указује на исти објекат Konj.
Ако mojSisar указује на неки други тип Sisari, оператор as уместо тога враћа вредност null:

```
Konj mojKonj = new Konj (...);
Sisari mojSisar = mojKonj;
//...
Konj mojKonjPonovo = mojSisar as Konj; //u redu - mojSisar je Konj
//...
Kit mojKit = new Kit (...);
mojSisar = mojKit;
//...
mojKonjPonovo = mojSisar as Konj; //vraca null - mojSisar je Kit
```

<u>Декларисање нових метода</u>
Често се дешава да програмер покушава поново да искористи назив за методу која већ постоји у класи у хијерархији наслеђивања.
Ако се деси да основна класа и изведена класа декларишу две методе који имају исти потпис, добија се упозорење када се апликација компајлира (потпис методе: назив, број и типови параметара).
Метода у изведеној класи маскира (или сакрива) методу у основној класи која има исти потпис.
Нпр, ако се компајлира следећи код, компајлер генерише поруку са упозорењем, у којој пише да метода Konj.Glas сакрива наслеђену методу Sisar.Glas:

```
class Sisari
{
	//...
	public void Glas()
	{
		//pretpostavka je da se svi sisari oglasavaju
	}
}
class Konj: Sisari
{
	//...
	public void Glas()
	{
		//konji se drugacije oglasavaju od ostalih sisara
	}
}
```

Иако ће се код компајлирати и покренути, ово упозорење је озбиљно; ако се још једна класа изводи из класе Konj и позива методу Glas, можда се очекује да буде позвана метода која се примењује у класи Sisari.

Међутим, метода Glas у класи Konj сакрива методу Glas у класи Sisari, па ће бити позвана метода Konj.Glas.

Најчешће, таква подударност је одличан извор забуна и било би добро да се размисли о промени назива метода како би се избгли проблеми.

Али, ако је неопходно да две методе имају исти потпис, чиме се сакрива метода Sisari.Glas, упозорење се може уђуткати коришћењем службене речи new:

```
class Sisari
{
	//...
	public void Glas()
	{
		//...
	}
}
class Konj: Sisari
{
	//...
	new public void Glas()
	{
		//...
	}
}
```

Овакво коришћење new не мења чињеницу да су ове две методе се потпуно независне и да се сакривање и даље дешава али се тиме само искључује упозорење.

Задаци за самосталан рад

1. Написати програм са основном класом Sisari и дифолтним конструктором који исписује унети параметар. Потклаcа Konj користи дифолтни конструктор основне класе и модификује свој дифолтни конструктор двоструким исписом унетог параметра.

2. Написати прохграм који доказује да метода истог имена Glas и у основној класи Sisari и у потклаcи Konji сакрива методу у основној класи.

3. Написати програм којим се решава проблем давања упозорења на процес сакривања методе из претходног задатка.