**Apstraktna klasa**

Svaka klasa koja sadrži ključnu reč ***abstract*** i koja ima najmanje jednu apstraktnu metodu se naziva apstraktna klasa. Apstraktna metoda je metoda koja sadrži ključnu reč abstract i koja nema ni telo ni implementaciju. Apstraktna klasa ne može da se instancira dok svaka njena apstraktna metoda mora da bude nadjačana u svakoj ne apstraktnoj klasi koja nasleđuje apstraktnu klasu. To jednostavno znači da apstraktnu klasu i njene apstraktne metode koristite kad niste sigurni kakva će biti implementacija neke metode ali vam je ta metoda neophodna. Apstraktna klasa je u svakom slučaju nepotpuna klasa.

**Koje su još stvari bitne za apstraktnu klasu?**

Najbitnija stvar kod apstraktnih klasa je da se one koriste kada generalizacijom ne možete da obezbedite potpunu funkcionalnost drugih klasa jer se njihova funkcionalnost realizuje na drugačije načine. Apstraktne klase se koriste i za bezbednosne svrhe da onemoguće inicijalizaciju klase. Za razliku od statičkih klasa koji ne mogu da se nasleđuju, apstraktna klasa mora da se nasledi. Znači za statičku klasu se podrazumeva da je ***sealed*** i bez ključne reči ***sealed*** dok apstraktna klasa ne može da sadrži ključnu reč ***sealed*** jer je ***abstract*** kontradiktorno od ***sealed***. Apstraktne metode su automatski virtualne ali ako koristite ključnu reč ***virtual*** dobićete grešku. U [***C#***](http://www.manuelradovanovic.com/2015/11/uvod-u-c-programski-jezik.html) programskom jeziku ključnu reč ***virtual*** ne možete koristiti ni kod statičnih metoda, nadjačanih metoda i privatnih metoda. Kad nasledite apstraktnu klasu vi morate obezbediti funkcionalnost za svaku apstraktnu metodu u apstraktnoj klasi.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Klasa Figura**

using System;

using System.Collections.Generic;

using System.Text;

namespace ConsoleApp24

{

abstract class Figura

{

private int centarX, centarY;

private string ime\_figure;

private string boja\_figure;

public Figura(string i, int x, int y, string b)

{

centarX = x;

centarY = y;

ime\_figure = i;

boja\_figure = b;

}

public string Ime()

{

return ime\_figure;

}

public void PomeriZa(int dx, int dy)

{

centarX += dx;

centarY += dy;

}

public virtual void Ispisi()

{

Console.WriteLine("Objekat: {0}", ime\_figure);

Console.WriteLine("centar: {0}, {1}", centarX, centarY);

Console.WriteLine("boja: {0}", boja\_figure);

}

public abstract double Povrsina();

}

}

**Klasa Krug**

using System;

using System.Collections.Generic;

using System.Text;

namespace ConsoleApp24

{

class Krug :Figura

{

private int r\_kruga;

public Krug(string i, int x, int y, string b, int r) :base(i, x, y, b)

{

r\_kruga = r;

}

public override void Ispisi()

{

base.Ispisi();

Console.WriteLine(" poluprecnik: {0}", r\_kruga);

}

public override double Povrsina()

{

return Math.PI \* Math.Pow(r\_kruga, 2);

}

}

}

**Klasa Pravougaonik**

using System;

using System.Collections.Generic;

using System.Text;

namespace ConsoleApp24

{

class Pravougaonik :Figura

{

private int duzina, sirina;

public Pravougaonik(string i, int x, int y, string b, int d, int s) :base (i, x,y,b)

{

duzina = d;

sirina = s;

}

public override void Ispisi()

{

base.Ispisi();

Console.WriteLine("dimenzije: {0}, {1}", sirina, duzina);

}

public override double Povrsina()

{

return sirina \* duzina;

}

}

}

**Glavni program**

using System;

namespace ConsoleApp24

{

class Program

{

static void Main(string[] args)

{

// Figura f = new Figura("figura", 5, 3, "plava");

// f.Ispisi();

Krug k = new Krug("krug", 10, 5, "crvena", 1);

k.Ispisi();

Console.WriteLine("Povrsina kruga: {0}", k.Povrsina());

Pravougaonik p = new Pravougaonik("pravougaonik", 2, 3, "zelena", 5, 8);

p.Ispisi();

Console.WriteLine("Povrsina pravougaonika: {0}", p.Povrsina());

}

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Interface**

Klasa ne može da nasleđuje istovremeno više osnovnih klasa. Ali zato može da implementira više interfejsa, istovremeno. Dok osnovna klasa definiše šta ona u stvari predstavlja, interfejs definiše šta klasa radi. Interfejs ne može da sadrži polja. Na taj način osiguravate da klasa koja je u interakciji kroz određeni interfejs nema pristup internim podacima objekta. Interfejsi jesu slični apstraktnim klasama, samo što ne mogu da sadrže ništa osim navedenih potpisa i nije retkost da se interfejsi koriste zajedno sa apstraktnim klasama da bi se gradila programerska radna okruženja koja se proširuju. Interfejs, kao i apstraktna klasa se ne može instancirati.

Implementacija interfejsa se u potpunosti prepušta implementirajućoj klasi. Svaki član interfejsa za razliku od klasa se podrazumeva da je javni, svaki član mora biti javni i svaki član se mora implementirati u istoj implementirajućoj klasi. Ali najvažnija osobina interfejsa je da klase mogu implementirati više interfejsa kao i što interfejs može da poziva druge interfejse. Kada se iz jednog interfejsa pozivaju drugi interfejsi, onda se to naziva ***re-implementation***.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

using System;

using System.Collections.Generic;

using System.Text;

namespace ConsoleApp21

{

interface Interface1

{

int GetInt();

double GetPi();

int Square(int x);

double Sqrt(double x);

}

public interface IMojInterface

{

double Sqrt2(double x);

}

public interface InterfaceSetGet

{

double MyPi

{

get; set;

}

}

}

**Klasa**

using System;

using System.Collections.Generic;

using System.Text;

namespace ConsoleApp21

{

class Class1 : Interface1, IMojInterface, InterfaceSetGet

{

private double doublePi = 6.28;

public double MyPi

{

get { return doublePi; }

set { doublePi = value; }

}

public int GetInt()

{

return 25;

}

public double GetPi()

{

return Math.PI;

}

public int Square(int x)

{

return (x \* x);

}

public double Sqrt(double x)

{

return Math.Sqrt(x);

}

public double Sqrt2(double x)

{

return Math.Sqrt(x);

}

}

}

**Glavni program**

using System;

namespace ConsoleApp21

{

class Program

{

static void Main(string[] args)

{

Class1 c = new Class1();

Console.WriteLine(c.GetPi());

Console.WriteLine(c.GetInt());

Console.WriteLine("dupla vrednost za pi:"+ c.MyPi);

Console.WriteLine("nova vrednost za pi=" + c.MyPi);

}

}

}

Razlika između apstraktne klase i interfejsa: https://www.ucim-programiranje.com/2019/02/interfejs-apstraktna-klasa-razlike-primer/

**Generička klasa**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Generičke klase omogućavaju parametrizovanje tipova, to znači da se može definisati klasa ili metoda sa generičkim (opštim) tipom. Jedna od postojećih generičkih klasa u C#-u koju smo koristili jeste List. Pomoću ove generičke klase možemo da kreiramo listu brojeva, stringova, objekata, struktura

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Karakteristika ovako definisane klase Test ima osobinu da može čuvati bilo koji tip iz programskog jezika C#. Sa kojim tipom će klasa test raditi, navodi se prilikom njenog instanciranja.

// Use using to declare namespaces and functions we wish to use

using System;

using System.Collections.Generic;

// Give our code a custom namespace

namespace ConsoleApp1

{

class Program

{

public class GenericClass<T>

{

private T msg;

public void setMsg(T msg0) { msg = msg0; }

public T value

{

get

{

return this.msg;

}

set

{

this.msg = value;

}

}

public void genericMethod(T name, T location)

{

Console.WriteLine("{0}", msg);

Console.WriteLine("Name: {0}", name);

Console.WriteLine("Location: {0}", location);

}

}

static void Main(string[] args)

{

GenericClass<string> gclass = new GenericClass<string>();

gclass.setMsg("Welcome");

gclass.genericMethod("Darko Popovic", "Beograd");

GenericClass<int> gclass2 = new GenericClass<int>();

gclass2.setMsg(3);

gclass2.genericMethod(222, 3333);

GenericClass<int> gclass3 = new GenericClass<int>();

gclass3.value = 42;

Console.WriteLine(gclass3.value);

//Kao da smo pisali ovo:

// public class GenericClass

//{

// public string msg;

// public void genericMethod(string name, string location)

// {

// Console.WriteLine("{0}", msg);

// Console.WriteLine("Name: {0}", name);

// Console.WriteLine("Location: {0}", location);

// }

//}

}

/\*

\* Generic means the general form, not specific. In C#, generic means not specific to a particular data type.

C# allows you to define generic classes, interfaces, abstract classes, fields, methods, static methods, properties, events, delegates, and operators using the type parameter and without the specific data type. A type parameter is a placeholder for a particular type specified when creating an instance of the generic type.

A generic type is declared by specifying a type parameter in an angle brackets after a type name, e.g. TypeName<T> where T is a type parameter.

A generic class increases the reusability. The more type parameters mean more reusable it becomes. However, too much generalization makes code difficult to understand and maintain.

A generic class can be a base class to other generic or non-generic classes or abstract classes.

A generic class can be derived from other generic or non-generic interfaces, classes, or abstract classes.

\*/

}

}