Seminář PRG
20. hodina - 7.2.2025

Gymnázium Voděradská 2024/2025
Jan Borecký

Dnešní téma

- Abstraktní třídy
- Interface
- Zadání 3. úkolu

Abstraktní třídy

Abstraktní třídy

- Abstraktní třídy využijeme jako “šablonu” pro třídy
- Vezměme si využívaný příklad Animal a jeho funkci AnimalNoise()

```
abstract class Animal
{
	//všimněte si, že AnimalNoise nemá tělo/implementaci
	public abstract void AnimalNoise();
}

class Dog : Animal
{
	//override už známe
	public override void AnimalNoise()
	{
		Console.WriteLine(“Woof!”);
	}
}
```

Abstraktní třídy

- Abstraktní funkci a její override ekvivalent ve třídách, které dědí, už známe. Jaký je tedy rozdíl?

- Abstraktní třídy nelze instancovat!

//dostaneme error, že se snažíme instancovat abstraktní třídu
Animal newAnimal = new Animal();

- Využití abstraktních tříd/funkcí:
  - Neexistuje smysluplná výchozí implementace
  - Každá třída, která dědí, musí metodu implementovat jinak
  - Chceme zamezit možnosti instancování oné třídy

Interface

Interface (rozhraní)

- Co když potřebujeme sdílet určité chování mezi třídami, které si nejsou blízko v hierarchii? (např. letadlo i pták létají)
- Co když potřebujeme dědit od více tříd? (lze totiž jenom od jedné)

```
interface IFly
{
    //výchoze public a abstract, nemusím to psát
    void Fly();
}
```

- Je zvykem psát první písmeno názvu velkým měkkým I

Interface

- Využití interface (hlavně toho, že lze dědit z více naráz):

```
interface IGetDamage
{
	void GetDamage(); //výchoze public a abstract, nemusím to psát
}

interface IAttack
{
	void DealDamage();
}
```

- Když budeme dělat třídu pro enemáky, chceme, aby uměli jak dostat damage, tak ho i dát

```
class Enemy : IGetDamage, IAttack
```

- Když budeme dělat třídu pro npc/civilisty, chceme, aby uměli jenom dostat damage

```
class Npc : IGetDamage //Ale ne IAttack
```

Interface

- Většinou kombinujeme s abstraktními třídami (popř. klasickými třídami)

- Interfacy doplňujeme vlastnosti, které chceme mít izolované

- Poctivé rozdělení do abstraktních tříd a interfaců nám zaručuje flexibilitu kódu a většinou i přehlednost

Zadání 3. úkolu

- Hra v konzoli - hráč ovládá hru příkazy, které píše do konzole
- Může to být detektivka, rpg, poker, vaše oblíbená kampaň Dračáku... Kreativitě se meze nekladou :)
- Promyslete, co byste rádi dělali a sepište design dokument, který mi pošlete na Discord nebo na mail
- Hodnocení
  - Design dokument
  - Coding style
  - Rozsah a funkčnost celé hry (individuální podle toho, s čím přijdete)
  - **Povinnost využít třídy!** (ideálně využít abstrakci, interfacy atd.)
- Deadliny:
  - 13.2. 23:59 - Design dokument
  - 2.3. 23:59 - Odevzdání kompletní hry

Design dokument

Co by měl design dokument obsahovat?

- Popis příběhu / světa / situace, ve které hra je
- Game loop - hlavní herní mechanika, kterou hráč hru hraje
- Návrh tříd a jejich proměnných a funkcí
  - Funkce nerozepisujte
  - Neřešte konstruktor
  - Popište, jak mezi sebou budou jednotlivé třídy interagovat
- Harmonogram vaší práce na hře
  - Popište, co musíte udělat jako první, co bude pak následovat, co bude následovat jako třetí atd.
  - Konkrétní dny a časy, kdy se tomu budete věnovat, nejsou relevantní

Děkuji za pozornost

Zpětná vazba:
https://forms.gle/9PdobjBq7vDnA41n6

Kontakt:
Mail - honza.borecky@seznam.cz
Discord - yeenya (Yeenya#6930)