UNIVERSITY SIMULATOR

PROJEKT IZ OBJEKTNO-ORIJENTIRANOG PROGRAMIRANJA

Lucija Bročić, lipanj 2018.

UNIVERSITY SIMULATOR

- University Simulator je igrica simulacije (simulation video game) u kojoj igrač kao student kroz 10 dana mora proći ispite, tj. poboljšati svoje sposobnosti te do 5.-tog dana doći na 50%, a do zadnjeg, 10.-tog dana na 90% vrijednosti za “knowledge”, “charisma” i “fitness”, pri su ograničavajući faktori “energy” i “stress level”.
- U daljnoj dokumentaciji ću predstaviti kako sam kroz ovaj projekt implementirala osnovne koncepte i ideje objektno orijentiranog programiranja.

PROCEDURALNO I OBJEKTNO-ORIJENTIRANO PROGRAMIRANJE

- Proceduralno programiranje koristi procedure, blokove naredbi koje su grupirane u smislene cjeline, te podatke sprema u varrijable, lokalne (unutar procedure) ili globlane.
- OOP je proširenje proceduralnog programiranja.
- Javljaju se objekti koji su konceptualno slični konkretnim objektima u stvarnom svijetu, mogu sadržavati vlastite varijable i metode te atribute.

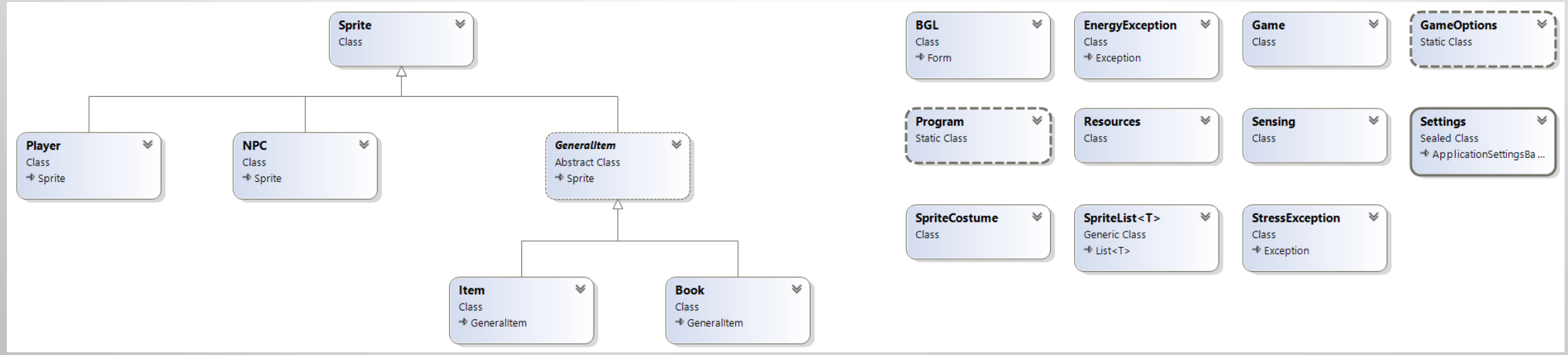
KLASA I OBJEKT

- Osnova OOP-a su objekti i interakcija među objektima
- Klasa je obrazac u kojem su spremljena različita svojstva i metode objekta. Po tom obrascu stvaramo pojedine objekte, pri čemu njihova svojstva poprimaju različite vrijednosti.
- Objekte instanciramo pomoću ključne riječi new te konstruktora, public metode u klasi.

Klase u projektu:

- Apstraktna klasa GeneralItem
- Izvedene klase Item i Book
- Klase Player i NPC
- Klase izvedene iz Exception klase – EnergyException i StressException
- Već postojeće klase u OTTER-u (Sprite, Sensing, GameOptions…)

DIJAGRAM KLASA U PROJEKTU



APSTRAKTNE I STATIČKE KLASE

- GeneralItem je apstraktna klasa. Apstraktne klase se ne mogu instancirati, ali možemo izvoditi klase iz njih
- GeneralItem okuplja svojstva koja imaju i Item i Book. No, Item i Book se razlikuju po dvjema svojstvima – Item troši vrijeme, ali ne novac, dok Book troši novac, a ne vrijeme
- Statičke klase su klase koje nije potrebno instancirati da bi se pristupilo njihovim svojstvama i metodama.U projektu je to primjerice klasa GameOptions

```
player.X = 0;
player.Y = GameOptions.DownEdge/2 - player.Heigth;
```

Instance ovih klasa su:

- Item: desk, bed, table, bookcase, tv, karaoke, dumbbell, computer, workdesk
- Book: bKnowledge, bCharisma, bFitness, bStress
- NPC: professor, trainer, boss, friend1, friend2
- Player: player

ČETIRI STUPA OOP-A

- Enkapsulacija (učahurivanje)
- Apstrakcija
- Nasljeđivanje
- Polimorfizam

ENKAPSULACIJA (UČAHURIVANJE)

- Skrivanje i povezivanje unutrašnjih elemenata nekog objekta u povezanu i smislenu cjelinu tako da unutrašnji ustroj nije vidljiv vanjskom svijetu
- U projektu, enkapsulacija se vidi u ograničavanju prava pristupa određenim elementima objekta
- private polja unutar klase enkapsuliramo public svojstvima te preko metoda pristupa (get i set) unutar tih svojstava pristupamo poljima

PRIMJER ENKAPSULACIJE

```
private int knowledge;
public int Knowledge
{
    get { return knowledge; }
    set
    {
        if (value < 0)
            knowledge = 0;
        else if (value > 100)
            knowledge = 100;
        else
            knowledge = value;
    }
}
```

PRAVA PRISTUPA ELEMENTIMA

- Mogu se definirati prilikom deklaracije korištenjem ključnih riječi:
- public – element vidljiv svima, i izvan klase u kojoj je deklariran
- private – element vidljiv samo unutar klase u kojoj je deklariran
- protected – element vidljiv u klasi u kojoj je deklariran te svim klasama izvedenim iz početne

APSTRAKCIJA

- Slično enkapsulaciji
- Enkapsulacija služi zaštiti podataka od vanjskog svijeta
- Služi skrivanju nepotrebnih dijelova, stvarajući manje kompleksan sustav
- U projektu je apstraktna klasa GeneralItem primjer apstrakcije

NASLJEĐIVANJE

- Iz osnovne klase možemo izvesti podklase postupkom nasljeđivanja
  - Osnovna klasa - klasa koja služi kao osnova za nasljeđivanje
  - Izvedena klasa – klasa koja nasljeđuje elemente od osnovne i sadrži nove
- U projektu je, primjerice, GeneralItem osnovna klasa (iako je i ona podklasa od klase Sprite), a klase Item i Book izvedene klase

```
class Item : GeneralItem
```

```
class Book:GeneralItem
```

NEW I OVERRIDE

- Ako nam neke metode ili svojstva iz osnovne klase nisu prikladni za upotrebu u izvedenoj klasi, onda ih prilagođavamo (premošćujemo)
- Koristimo ključne riječi override i new
  - new – koristi se nova implementacija umjesto one u osnovnoj klasi, ali ako instanciramo objekt izvedene klase kao objekt osnovne, koristi se svojstvo iz osnovne klase
  - override – koristi se posljednje definirana implementacija

OVERRIDE PRIMJER

```
public override int X
{
    get { return base.X; }
    set {
        if (value < GameOptions.LeftEdge-GameOptions.SpriteWidth/2)
            base.X = GameOptions.RightEdge-GameOptions.SpriteWidth-50;
        else if (value > GameOptions.RightEdge-GameOptions.SpriteWidth)
            base.X = -GameOptions.SpriteWidth / 2;
        else
            base.X = value;
    }
} //override-ano svojstvo za X koordinatu
```

POLIMORFIZAM

- Svojstvo metode da različito djeluje ovisno o kontekstu
- Primjer: metoda SetHeading u klasi Sprite
- SetHeading može primiti DirectionsType podatak, ali i podatak tipa int

```
/// <summary> Postavlja smjer.
public void SetHeading(DirectionsType heading)...

/// <summary> Metoda koja pomiče lika u 4 osnovna smjera: lijevo, desno, gore i  ...
public void MoveSimple(int steps)...

/// <summary> Postavlja smjer lika.
public void SetHeading(int newDirectionAngle)... //Setdir
```

OVERLOAD

- Preopterećenje metoda je svojstvo objektno-orijentiranog programiranja koje omogućava postojanje metoda s istim imenom, ali različitim svrhama
- To se postiže promjenom broja parametara i/ili promjenom tipa podataka
- Primjer: konstruktor klase Item

```
class Item : GeneralItem
{
    private int timeMin;
    public int TimeMin...

    public Item(string s, int x, int y, int knowledge, int charisma, int fitness, int stress, int energy, int time)...

    public Item(string s, int x, int y) //overload konstruktor...
}
```

UPRAVLJANJE IZNIMKAMA

- Iznimka je svako pogrešno ili neočekivano stanje prilikom izvršavanja programa
- Sve iznimke su objekti klase Exception
- Iznimkama upravljamo preko try, catch i finally blokova
- U try bloku pišemo kod koji želimo da se izvrši, catch blok hvata iznimke koje se jave prilikom izvršavanja koda u try bloku, a finally blok služi za kod koji se svakako mora izvršiti

- U projektu postoje dvije klase izvedene iz klase Exception: StressException i EnergyException

```
class EnergyException : Exception
{
    private static string message = "You don't have enough energy!";
    public EnergyException()...
}

class StressException: Exception
{
    private static string message = "You are too stressed!";
    public StressException()...
}
```

```
if (player.TouchingSprite(desk) && sensing.KeyPressed("T"))
{
    Wait(0.5);

    try
    {
        player.Energy += desk.EnergyChange;
        player.Knowledge += desk.KnowledgeChange;
        player.Stress += desk.StressChange;
        TimerUpdate(desk.TimeMin);
    }
    catch (EnergyException en)
    {
        Wait(0.5);
        MessageBox.Show(en.Message);
    }
    catch (StressException se)
    {
        Wait(0.5);
        MessageBox.Show(se.Message);
    }
```

DOGAĐAJI

- Događaj je reakcija na neku pojavu unutar programa
- Kao odgovor na događaj definiramo posebne metode za upravljanje događajima (event handler)
- Primjer: pritisak tipke na tipkovnici ili mišu

```
private void buttonStart_Click_1(object sender, EventArgs e)
{
    panelStart.Visible = false;
    panelOpening.Visible = false;

    CreatePlayer();
    SetupGame();
}

private void buttonExitGame_Click(object sender, EventArgs e)...

private void buttonStart_Click(object sender, EventArgs e)...
private void buttonLoadGame_Click(object sender, EventArgs e)...
```

Event handler - metode za upravljanje događajima u klasi Sensing

```
/// <summary> Provjerava je li tipka koja je poslana kao parametar pritisnuta.
public bool KeyPressed(string keyName)
{
    if (KeyPressedTest && Key == keyName)
    {
        Game.WaitMS(20);
        return true;
    }
    else
        return false;
}

/// <summary> Provjerava je li tipka koja je poslana kao parametar pritisnuta.
public bool KeyPressed(Keys key)
{
    if (KeyPressedTest && Key == key.ToString())
    {
        Game.WaitMS(20);
        return true;
    }
    else
        return false;
}
```

DELEGATI

- Delegat je objekt koji sadrži referencu na metodu
- Omogućuje da reference na metodu koristimo kao ulazni argument neke druge metode
- Primjer deklaracije:

```
delegate string PrimjerDelegat(int x);
```

```
public string PrimjerMetoda(int broj)
{
    return broj.ToString();
}
```

HVALA NA POZORNOSTI!