Operator Overload

Peter Levinsky IT Roskilde

05.04.2021

Support foreach-loops

- Making a Class Enumerable
  – by implementing **IEnumerable**<type-of--value> i.e. two methods

```
public IEnumerator<SomeType> GetEnumerator()
{
     yield .. break;
}

IEnumerator IEnumerable.GetEnumerator()
{
     return GetEnumerator();
}
```

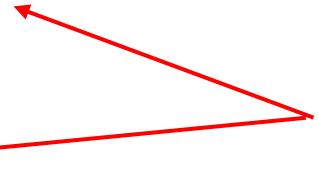
Overload

- Four categories:
  - Indexer (i.e. [])
  - +,–, *, / … Operator
  - <= and >= Operator
  - == and != Operator

Overload - Indexer (i.e. [])

```
public class NPC
{
   private Dictionary<NPCStateTypes, int> State;

   public NPC()
   {
      State = new Dictionary<NPCStateTypes, int>();
      SetDefaultValues();
   }

   public void SetStateValue(NPCStateTypes stateType, int
   value)
   {
      State[stateType] = value;
   }

   public int GetStateValue(NPCStateTypes stateType)
   {
      return State[stateType];
   }

   private void SetDefaultValues() { … }
}
```

```
public enum NPCStateTypes
{
   hungry, rested, aggressive, fear, gullible
}
```



Wish to use [] Instead of GetStateValue, SetStateValue

Overload - Indexer (i.e. [])

- The implementation:

meaning this class

define the brackets + what's inside

```
public int this[NPCStateTypes stateType]
{
    get { return State[stateType]; }
    set { State[stateType] = value; }
}
```

Overload: +,-,*,/,++ and -- Operator

- Example:

```
public static Time operator +(Time tA, Time tB)
{
…     // this add two time objects and return a new
}
```

Call:

```
Time timeC = timeA + timeB;
```

The general definition

```
public static TReturn operator operatorGoesHere(TA opA, TB opB)
{
      …
}
```

Overload: +,-,*,/,++ and -- Operator

- Example variation:

```
public static Time operator *(int val, Time t)
{
   int totalMin = (t.Hours * 60 + t.Minutes) * val;
   return new Time(totalMin / 60, totalMin % 60);
}
```

Call:

```
Time timeMult3A = 3 * timeA; // OK
```

but

```
Time timeMultA3 = timeA * 3; // Error
```

```
public static Time operator *(Time t, int val) // reverse parameter order
{
      return val * t;
}
```

Overload == and != Operator

If overload == also overload != ☺

```
public static bool operator ==(Time tA, Time tB)
{
   return (tA.Length == tB.Length); // tA.Equals(tB)
}

public static bool operator !=(Time tA, Time tB)
{
   return !(tA == tB);
}
```

BUT this implementations do not work – risk for NullReferenceException! DO THIS!

1. First override Equals appropriately (i.e. also checking for null)
2. As a consequence of overriding Equals, we must also override the method GetHashCode from Object.
3. Now override == and != by using Equals

Overload: <= and >= Operator (< and >)

If overload >= also overload <= ☺

```
public static bool operator >=(Time tA, Time tB)
{  // ToDo check for null
   return (tA.Length >= tB.Length);
}

public static bool operator <=(Time tA, Time tB)
{
   return (tA.Length <= tB.Length);
}
```

That’s it

- **Trænings OPGAVER: OOP.4.1, OOP.4.2**
- **OOP.4.5**

- **Samt Obligatorisk Aktivitet**