Recursion & Scoping of Variables II

Software Engineering 1

Dr Swen E. Gaudl

University of Plymouth 2022

COMP10000

Prep For Thursday:

- Looking into the Crawler and Handbook
- Come with questions!

COMP1000 Agenda This Week:

- Scoping Functionality/Variables
- Recursion
- Q&A

Scoping

- Defines Access/ Lifetime of Variables
- Relates to code blocks { }
- Priority local -> Object -> Class -> Global

```
public class Monster
{
    protected float health = 100.0f;
    protected float damage = 5.0f;
    protected float armor = 100.0f;
    public static int VERSION = 0;

    public bool TakeDamage(int damage)
    {
        float health = health + armor;
    }
    public void DealDamage(Monster[] monsters)
    {}

    public static void Main()
    {
       if (Monster.VERSION > 0)
          Console.WriteLine("New Monster”+
             ” Class found");

     }

}
```

Scoping

```
public class Monster
{
        protected float health = 100.0f;
        protected float damage = 5.0f;
        protected float armor = 100.0f;
        public static int VERSION = 0;

        public bool TakeDamage(int damage)
        {
            float health = health + armor;
        }
        public void DealDamage(Monster[] monsters)
        {}

        public static void Main()
        {
           if (Monster.VERSION > 0)
              Console.WriteLine("New Monster”+
                 ” Class found");

         }

}
```

Scoping

```
public class Monster
{
    protected float health = 100.0f;
    protected float damage = 5.0f;
    protected float armor = 100.0f;
    public static int VERSION = 0;

    public void TakeDamage(int damage)
    {
       float health = this.health + armor;
    }
    public void DealDamage(Monster[] monsters)
    {
        int damage = 0;
        foreach (Monster monster in monsters)
            monster.TakeDamage(damage);
    }
}
```

Scoping

Further Reading:

- https://www.geeksforgeeks.org/c-sharp-types-of-variables/
- http://www.blackwasp.co.uk/CSharpVariableScopes.aspx
- https://www.tutorialspoint.com/scope-of-variables-in-chash

- Defines Access/ Lifetime of Variables
- Relates to code blocks { }
- Priority local -> Object -> Class -> Global

Recursion

- Algorithmic process
- When a calls a method/function directly/ indirectly calls itself
- Ideal for some iterative processes
- Similar to loops

- Notes:
  - Code can be more difficult to write
  - Will introduce complexity in readability
  - Not always ideal as it adds to the call stack

https://www.techiedelight.com/recursion-practice-problems-with-solutions/

Recursion

- Algorithmic process
- When a calls a method/function directly/ indirectly calls itself
- Ideal for some iterative processes
- Similar to loops

- Notes:
  - Code can be more difficult to write
  - Will introduce complexity in readability
  - Not always ideal as it adds to the call stack

```
public int SumUp(int steps)
{
    int result = 0;
    for (int i = steps; i >= 0; i--)
        result = result + i;

    return result;
}
```

Recursion

```
public int SumUp(int steps)
{
    int result = 0;
    for (int i = steps; i >= 0; i--)
        result = result + i;

    return result;
}
```

Recursion

- Notes:
  - Code can be more difficult to write
  - Will introduce complexity in readability
  - Not always ideal as it adds to the call stack

```
public int SumUp(int steps)
{
    int result = 0;
    for (int i = steps; i >= 0; i--)
        result = result + i;

    return result;
}
```

```
public int RecursiveSumUp(int steps)
{
    if (steps <= 0)
        return 0;

    return steps+RecursiveSumUp(steps-1);
}
```

Recursion

```
public int RecursiveSumUp(int steps)
{
    if (steps <= 0)
        return 0;

    return steps+RecursiveSumUp(steps-1);
}
```

Recursion

- Algorithmic process
- When a calls a method/function directly/ indirectly calls itself
- Ideal for some iterative processes
- Similar to loops

```
public int RecursiveSumUp(int steps)
{
    if (steps <= 0)
        return 0;

    return steps+RecursiveSumUp(steps-1);
}
```

```
public int RecursiveSumUpV2(int steps)
{
   return steps <=0 ? 0 : steps + RecursiveSumUpV2(steps - 1);
}
```

Next Time

- Commeting in C# where and how
- Interfaces and Abstract Classes