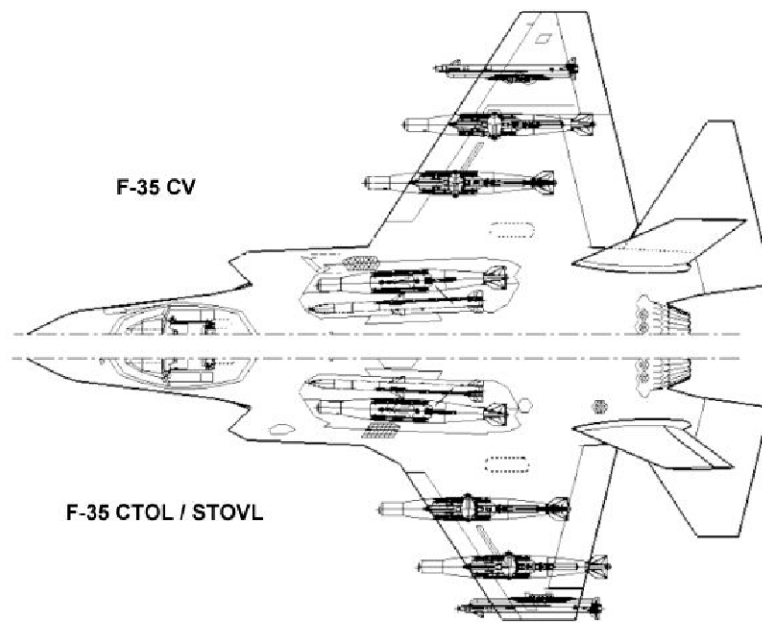


1. interfaces

Program to an interface, not an implementation

1. interfaces



.....

.....

.....

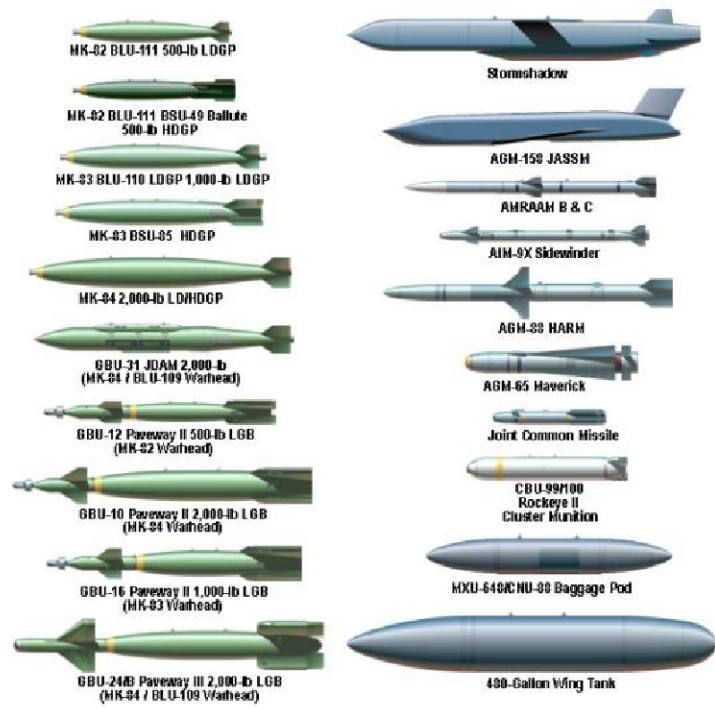
.....

.....

.....

.....

1. interfaces



.....

.....

.....

.....

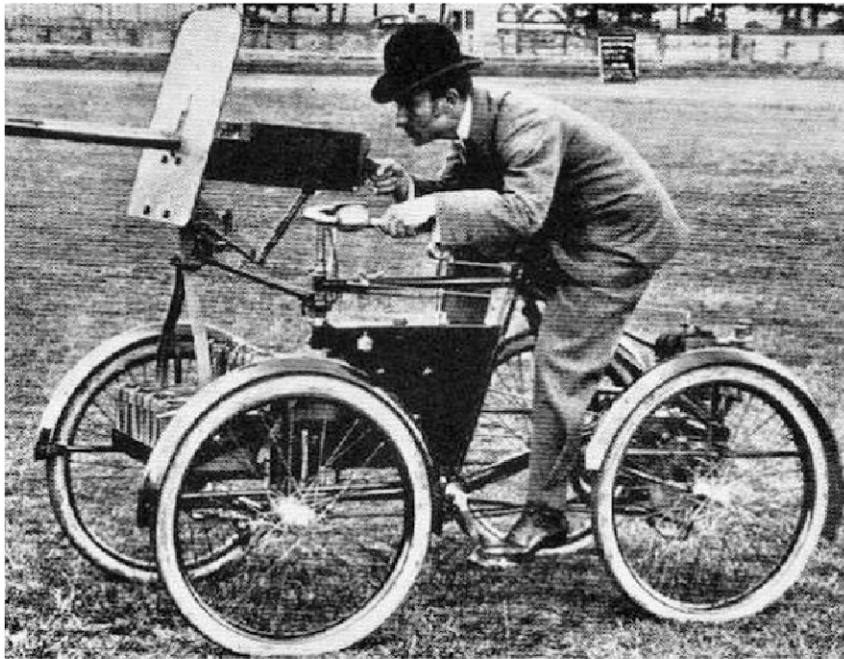
.....

.....

.....

.....

1. interfaces



.....

.....

.....

.....

.....

.....

.....

1. interfaces

```
using System.Collections.Generic;
using UnityEngine;

public class PlayerShip : MonoBehaviour
{
    private PlasmaRifle activeWeapon;

    void Start()
    {

    }

    void Shoot()
    {
        activeWeapon.shoot();
    }
}
```

```
.....
.....
.....
.....
.....
.....
.....
.....
```

1. interfaces

```
using System.Collections.Generic;
using UnityEngine;


public class PlayerShip
{
    private PlasmaRifle activeWeapon;

    void Start()
    {

    }

    void Shoot()
    {
        activeWeapon.shoot();
    }
}
```

tight coupling
implementatie



.....

.....

.....

.....

.....

.....

.....

.....

1. interfaces

```
using System.Collections.Generic;
using UnityEngine;

public class PlayerShip
{
    private PlasmaRifle activeWeapon;

    void Start()
    {

    }

    void Shoot()
    {
        activeWeapon.shoot();
    }
}
```

niet zomaar andere
wapens zoals in de F35

.....

.....

.....

.....

.....

.....

.....

.....

1. interfaces

```
public interface IWeapon
{
    void Shoot();
}
```

.....

.....

.....

.....

.....


.....

.....

.....

1. interfaces

niet een class
maar een interface



```
public interface IWeapon
{
    void Shoot();
}
```

.....

.....

.....

.....

.....

.....

.....

.....

1. interfaces

```
using UnityEngine;

public class LaserCannon : IWeapon {
    private float fireDelay;
    private float timeSinceLastShoot;
    private GameObject owner;
    public GameObject projectilePrefab;

    public LaserCannon(float _fireDelay, GameObject _owner){
        owner = _owner;
        fireDelay = _fireDelay;
    }

    public void Shoot(){
        if (Time.time > fireDelay + timeSinceLastShoot)
        {
            GameObject projectile = (GameObject)
GameObject.Instantiate(projectilePrefab, owner.transform.position,
Quaternion.identity);
            timeSinceLastShoot = Time.time;
        }
    }
}
```

```
.....
.....
.....
.....
.....
.....
.....
.....
```

1. interfaces

```
using UnityEngine;
```

```
public class LaserCannon : IWeapon {  
    private float fireDelay;  
    private float timeSinceLastShoot;  
    private GameObject owner;  
    public GameObject projectilePrefab;  
  
    public LaserCannon(float _fireDelay, GameObject _owner){  
        owner = _owner;  
        fireDelay = _fireDelay;  
    }  
  
    public void Shoot(){  
        if (Time.time > fireDelay + timeSinceLastShoot)  
        {  
            GameObject projectile = (GameObject)  
GameObject.Instantiate(projectilePrefab, owner.transform.position,  
Quaternion.identity);  
            timeSinceLastShoot = Time.time;  
        }  
    }  
}
```

nieuwe class
met interface

.....

.....

.....

.....

.....

.....

.....

.....

1. interfaces

```
using UnityEngine;

public class LaserCannon : IWeapon {
    private float fireDelay;
    private float timeSinceLastShoot;
    private GameObject owner;
    public GameObject projectilePrefab;

    public LaserCannon(float _fireDelay, GameObject _owner){
        owner = _owner;
        fireDelay = _fireDelay;
    }

    public void Shoot(){
        if (Time.time > fireDelay + timeSinceLastShoot)
        {
            GameObject projectile = (GameObject)
            GameObject.Instantiate(projectilePrefab, owner.transform.position,
            Quaternion.identity);
            timeSinceLastShoot = Time.time;
        }
    }
}
```

die verplicht de shoot
functie ondersteunt

.....

.....

.....

.....

.....

.....

.....

.....

1. interfaces

```
using System.Collections.Generic;
using UnityEngine;

public class PlayerShip : MonoBehaviour {
    private IWeapon activeWeapon;
    private List weapons;

    void Start(){
        weapons = new List { new MissileLauncher(0.5f, gameObject),
                             new LaserCannon(0.5f, gameObject) };
    }

    public void Control(){
        if (Input.GetKey(KeyCode.Space)){
            activeWeapon.Shoot();
        }
        if (Input.GetKey(KeyCode.LeftControl)){
            SetWeapon(weapons[0]);
        }
        if (Input.GetKey(KeyCode.RightControl)){
            SetWeapon(weapons[1]);
        }
    }

    private void SetWeapon(IWeapon _weapon){
        activeWeapon = _weapon;
    }
}
```

.....

.....

.....

.....

.....

.....

.....

.....

1. interfaces

```
using System.Collections.Generic;
using UnityEngine;

public class PlayerShip :
    private IWeapon activeWeapon;
    private List weapons;

    void Start(){
        weapons = new List { new MissileLauncher(0.5f, gameObject),
                               new LaserCannon(0.5f, gameObject) };
    }

    public void Control(){
        if (Input.GetKey(KeyCode.Space)){
            activeWeapon.Shoot();
        }
        if (Input.GetKey(KeyCode.LeftControl)){
            SetWeapon(weapons[0]);
        }
        if (Input.GetKey(KeyCode.RightControl)){
            SetWeapon(weapons[1]);
        }
    }

    private void SetWeapon(IWeapon _weapon){
        activeWeapon = _weapon;
    }
}
```

het boeit PlayerShip niet wat
voor een wapen hij vuurt

.....

.....

.....

.....

.....

.....

.....

.....

1. interfaces

```
using System.Collections.Generic;
using UnityEngine;
```


```
public class PlayerShip : MonoBehaviour {
    private IWeapon activeWeapon;
    private List weapons;

    void Start(){
        weapons = new List { new MissileLauncher(0.5f, gameObject),
                             new LaserCannon(0.5f, gameObject) };
    }

    public void Control(){
        if (Input.GetKey(KeyCode.Space)){
            activeWeapon.Shoot();
        }
        if (Input.GetKey(KeyCode.LeftControl)){
            SetWeapon(weapons[0]);
        }
        if (Input.GetKey(KeyCode.RightControl)){
            SetWeapon(weapons[1]);
        }
    }

    private void SetWeapon(IWeapon _weapon){
        activeWeapon = _weapon;
    }
}
```

nu maak ik de wapens nog
hard aan in deze Class



.....

.....

.....

.....

.....

.....

.....

.....

1. interfaces

```
using System.Collections.Generic;
using UnityEngine;
```


```
public class PlayerShip : MonoBehaviour {
    private IWeapon activeWeapon;
    private List weapons;

    void Start(){
        weapons = new List { new MissileLauncher(0.5f, gameObject),
                             new LaserCannon(0.5f, gameObject) };
    }

    public void Control(){
        if (Input.GetKey(KeyCode.Space)){
            activeWeapon.Shoot();
        }
        if (Input.GetKey(KeyCode.LeftControl)){
            SetWeapon(weapons[0]);
        }
        if (Input.GetKey(KeyCode.RightControl)){
            SetWeapon(weapons[1]);
        }
    }

    private void SetWeapon(IWeapon _weapon){
        activeWeapon = _weapon;
    }
}
```

nog vrijer als je dat
vanaf buiten af doet



.....

.....

.....

.....

.....

.....

.....

.....

1. interfaces

Waar kun je aan denken bij interfaces?

- IMovable (iedereen met o.a. een Move() functie)
- IKillable (iedereen met o.a. een Die() / Kill() functie)
- IDamageable
- IUsable
- ISendable
-

.....

.....

.....

.....

.....

.....

.....

.....

1. interfaces

```
GetComponent<IsDamageable>().ApplyDamage(playerDamage);
```

[illegible]

1. interfaces

Abstract classes should primarily be used for objects that are closely related, whereas interfaces are better at providing common functionality for unrelated classes.

1. interfaces

Ook superhandig met de Factory pattern

1. interfaces

Oefening:

zie mail...

.....

.....

.....

.....

.....

.....

.....

.....