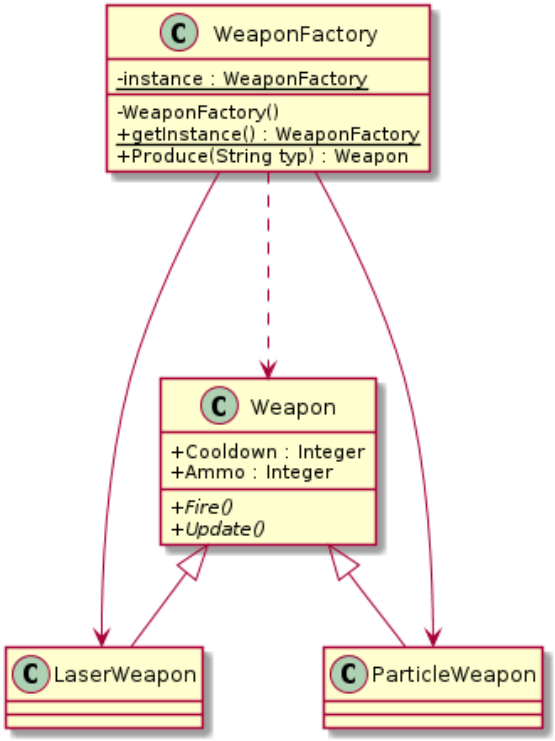
# Übungsblatt 2 (Lösung)

**Aufgabe 5** Eine Waffenfabrik errichten




**Aufgabe 6** Die Waffenfabrik implementieren

Die abstrakte Oberklasse für alle Waffen:

```
abstract class Weapon {
    public int Cooldown { get; protected set; }
    public int Ammo { get; protected set; }
    public abstract void Fire();
    public abstract void Update();
}
```

Die beiden konkreten Waffen:

```
class LaserWeapon : Weapon {
    bool firing;

    public override void Fire() {
        if (!firing)
            Console.WriteLine("Laser Strahl feuert ...");
        else
```

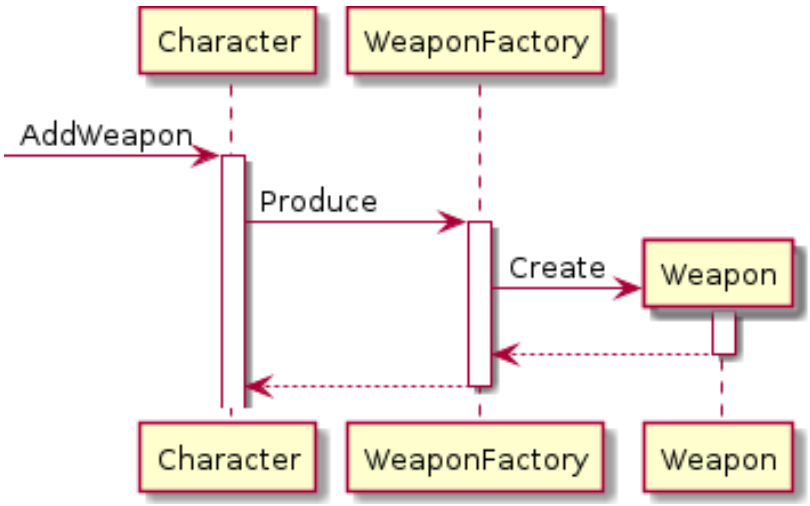
```
            Console.WriteLine("Laser Strahl feuert nicht mehr ...");
    }

    public override void Update() {
        if (firing)
            Console.Write(".");
    }
}

class ParticleWeapon : Weapon {
    public override void Fire() {
        if (Cooldown == 0) {
            if (Ammo > 0) {
                Console.WriteLine("Particle Weapon wurde gefeuert!!");
                Cooldown = 20;
                Ammo--;
            } else {
                Console.WriteLine("Particle Weapon hat keine Munition mehr!!");
            }
        }
    }

    public override void Update() {
        if (Cooldown > 0)
            Cooldown--;
    }
}
```

Die Waffenfabrik um die konkreten Waffen zu produzieren:

```
sealed class Waffenfabrik {
    static Waffenfabrik instance;
    Dictionary<String, Func<Weapon>> types;

    private Waffenfabrik() {
        types = new Dictionary<String, Func<Weapon>>();
        Register("laser", () => new LaserWeapon());
        Register("particle", () => new ParticleWeapon());
    }

    public static Waffenfabrik Instance {
        get { return instance ?? (instance = new Waffenfabrik()); }
    }

    void Register(String typ, Func<Weapon> constructor) {
        types.Add(typ.ToLower(), constructor);
    }

    public Weapon Produce(String typ) {
        typ = typ.ToLower();

        if (types.ContainsKey(typ))
            return types[typ]();

        return null;
    }
```

```
}
```

Das Sequenzdiagram mit dem Ablauf eines beispielhaften Aufrufs:




**Aufgabe 7** Charaktere bauen

Zunächst die Charakter-Klasse:

```
abstract class Character {
    public Weapon Weapon { get; set; }
    public Armor Armor { get; set; }
    public Ability Ability { get; set; }
}

class MarioCharacter : Character {
}

class YoshiCharacter : Character {
}
```

Die Klassen für Weapon, Armor und Ability:

```
abstract class Weapon {
    public static readonly Weapon None = new NoWeapon();
    sealed class NoWeapon : Weapon {
    }
}

abstract class Armor {
    public static readonly Armor None = new NoArmor();
    sealed class NoArmor : Armor {
    }
}

abstract class Ability {
    public static readonly Ability None = new NoAbility();
    sealed class NoAbility : Ability {
    }
}
```

Die Definition des Builders mit Implementierung für Yoshi:

```
interface IBuilder {
    IBuilder AddWeapon(Weapon weapon);
    IBuilder AddArmor(Armor armor);
    IBuilder AddAbility(Ability ability);
}

class YoshiBuilder : IBuilder {
    YoshiCharacter yoshi;

    public YoshiBuilder() {
        yoshi = new YoshiCharacter();
    }

    public YoshiCharacter Result {
        get { return yoshi; }
    }

    public void AddWeapon(Weapon weapon) {
        //Yoshi hat keine Waffe
        return this;
    }

    public void AddArmor(Armor armor) {
        //Yoshi hat auch keinen Armor
        return this;
    }

    public void AddAbility(Ability ability) {
        yoshi.Ability = ability;
        return this;
    }
}
```

Abschließend eine beispielhafte Verwendung inkl. Anbindung an eine Fabrik:

```
var build = new MarioBuilder();
build.AddArmor(Armor.None)
     .AddAbility(new FlyAbility())
     .AddWeapon(WeaponFactory.Produce("Fireball"));
var mario = build.Result;
```