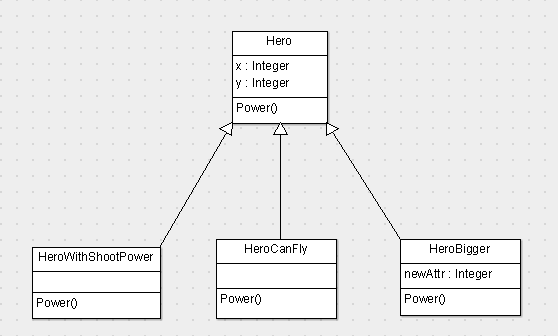
# Decorator patroon

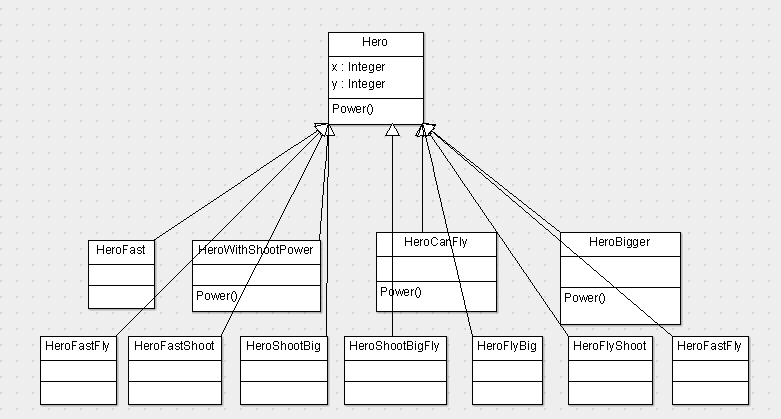
Problem:

You want to develop a platform game your hero can have multiple powers. He can shoot, fly, …

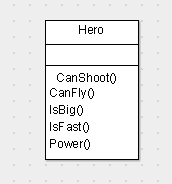
My first attempt:



But when you want to “decorate” your hero more and more, you have a design like:



Is this architecture managable? If you change the power method (abstract method in Hero class) you have to adapt it in every inherited class. So a better design might be:



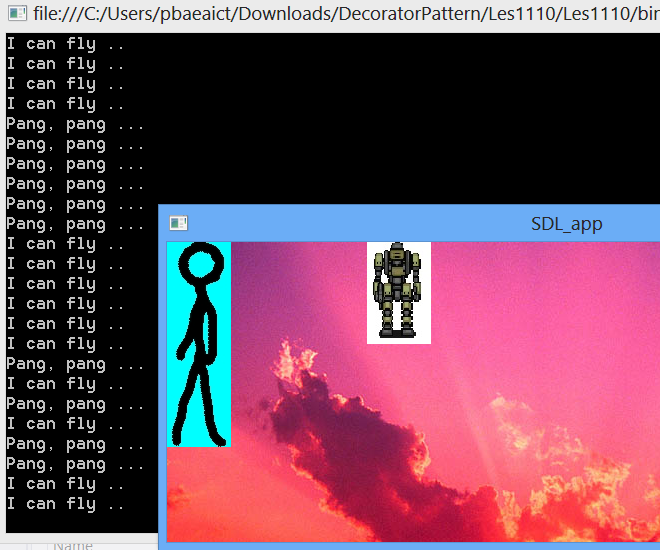
The power method figures out how you move, you can shoot by using booleans as CanShoot, CanFly, IsBig, ...

Disadvantage here is that you have to edit your base class every time you want to add new super-powers. Edit an existing class means testing again, because new code means new bugs…

So, let’s try to develop our class so you add new behavior without changing existing code. How do we accomplish this?

Let’s meet the decorator!

‘We are going to decorate’ our object with new functionality ( Hero will be bigger, he can shoot, )



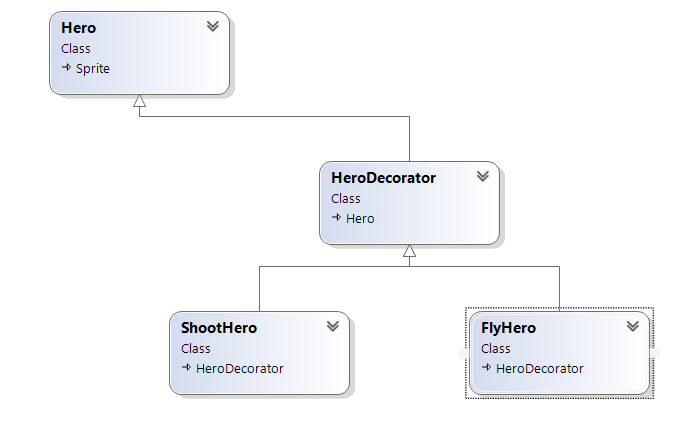
Design 1:

We tested our Hero class, so he runs left and right. Now I like to add shoot functionality, so I add som conditional logic ( if / else )

If ( key == key.space)

Hero.shoot();

Design 2: Don’t touch existing classes ( like the Hero class)



### Properties decorator pattern

Decorators have the same supertype as objects they decorate. More than one decorator can be wrapped over an object. This mean you can pass the decorated object instead of the original object

The Hero decorator heritates the Hero class, so decorators are the same type as the object the like to decorate (can shoot, can fly..)

Implementation DECORATOR PATTERN

De basis klasse Hero:

public class Hero : Sprite

{

public Hero(Surface vid)

{

Init(vid);

}

public virtual void power(SdlDotNet.Input.KeyboardEventArgs args)

{

//Original hero has no power..

}

De Hero Decorator:

class HeroDecorator : Hero

{

public HeroDecorator(Surface vid): base(vid)

{

}

public override void power(SdlDotNet.Input.KeyboardEventArgs args)

{

base.power(args);

}

}

class ShootHero: HeroDecorator

{

public Hero theHero;

public ShootHero(Surface vid, Hero h)

: base(vid)

{

theHero = h;

}

public override void power(SdlDotNet.Input.KeyboardEventArgs args)

{

if (args.Key == SdlDotNet.Input.Key.Space)

Console.WriteLine("Pang, pang ...");

theHero.power(args);

}

}

mHero = new Hero(mVideo, p);

mHero = new ShootHero(mVideo, mHero);

mHero = new FlyHero(mVideo, mHero);

//Terug naar origine hero!

((FlyHero)mHero).theHero = null;