2. Samenwerken (Implements)

- interface

3. Compositie ( 20-05-2019 )

Static variable:

1. Is er altijd precies 1x
2. Kan veranderen van waarden
3. Heeft een vaste plek. ( De plek staat wel vast. )
4. ( Globale variabele )
5. Kan overal benaderd worden via de Class

Wat we willen:

* Globale ‘variabele’ met object orientatie
  + Construtor
  + Maar er mag er maar 1 zijn.
  + Overal bereikbaar

Class beschermen tegen 2e keer aanmaken?

* Private constructor

Probleem: Wie hem de eerste keer aanmaken?

* Class zelf? (Maar die is er niet?)

Oplossing: De class een static method meegeven.

Class Singleton {

Private static instance: Singleton

Private Construtor {…}

Public static getInstance(): Singleton

If(!Singleton.instance)

{….}

Volgende week:

* 4x bom onder scherm = game over
* Status link boven bijwerken (kapotte gebouwen)
* Upgrades = extra ‘leven’
* Maak score singleton

Week 7: (oefening tentamen)

Code checken op fouten, ( oefenen voor tentamen )

Beschrijf waar de code gebruik maakt van:

* Inhheritance ( overerving )
  + De update functies
  + De ‘addChild’ in de constructor
* Composition ( relatie tussen 2 classes is een compositie )
  + ‘er is een relatie tussen 2 klassen’
    - Hij houd een lijst bij.
    - Kan 1 op 1 zijn.
    - Kan 1 op veel zijn.
    - Zonder dat het extend of overerft.
  + Zombie class met de movestrategy is een compositie.
  + Heeft een field ( attribuut )
  + Omdat je die andere classe nodig hebt om data te versturen.
    - “We hebben bijvoorbeeld bij de ‘Bom game’ de bom mee moeten sturen wanneer hij wordt geraakt.”
* Encapsulation
  + Toegankelijkheid van methods/fields
  + ‘waarom?’ gebruikt worden voor het afschermen van de interne werking.
    - Je wilt niet dat alles open staat voor publiek.
    - Makkelijker voor programmeur, hij ziet alleen dingen die nodig zijn.
    - Beheer, (beveiliging)
* Spelregel: Zo krap mogelijk.
  + Private
    - Field altijd
  + Protected
    - Field altijd
  + Public
    - Helaas in typescript nodig bij dubbele overerving.

Singelton:

* Overal beschikbaar ( zonder compositie )
* Beschermt tegen een 2e keer aanmaken.

Chicken:

* Iedereen wil/kan (un)subscriben by Chicken.
* Er mag maar 1 chicken zijn.

Fouten:

* Constructor public.
* Lijst van observer is static.

Polymorfisme:

* Gebruik maken van:

Parent/interface/abstracte class relatie

Dit is nog geen polymorfisme.

Gameobject

* + Chicken
  + Zombie
  + Phone

This.Chicken.update();

Nu wordt er wel gebruik gemaakt van polymorfisme.

Gameobject <- Update method welke alles update

* + Chicken
  + Zombie
  + Phone

If(chicken)Go.update() “acties worden afgehandeld op een hoger niveau”(polymorfisme).

Strategy pattern: (waarom?)

* Verwisselbare functionaliteit.

Observer: ( eigenlijk hetzelfde als compositie )

* Om andere objecten (verschillende classes) iets te laten weten.
* Om zichzelf aan andere objecten te subscriben of unsubscriben.
* Werkt niet zonder compositie.

Zombie – observer

Chicken – subject

Subject

Lijst van observers

Subscribe ( o )

Unsubscribe ( o )

Update();