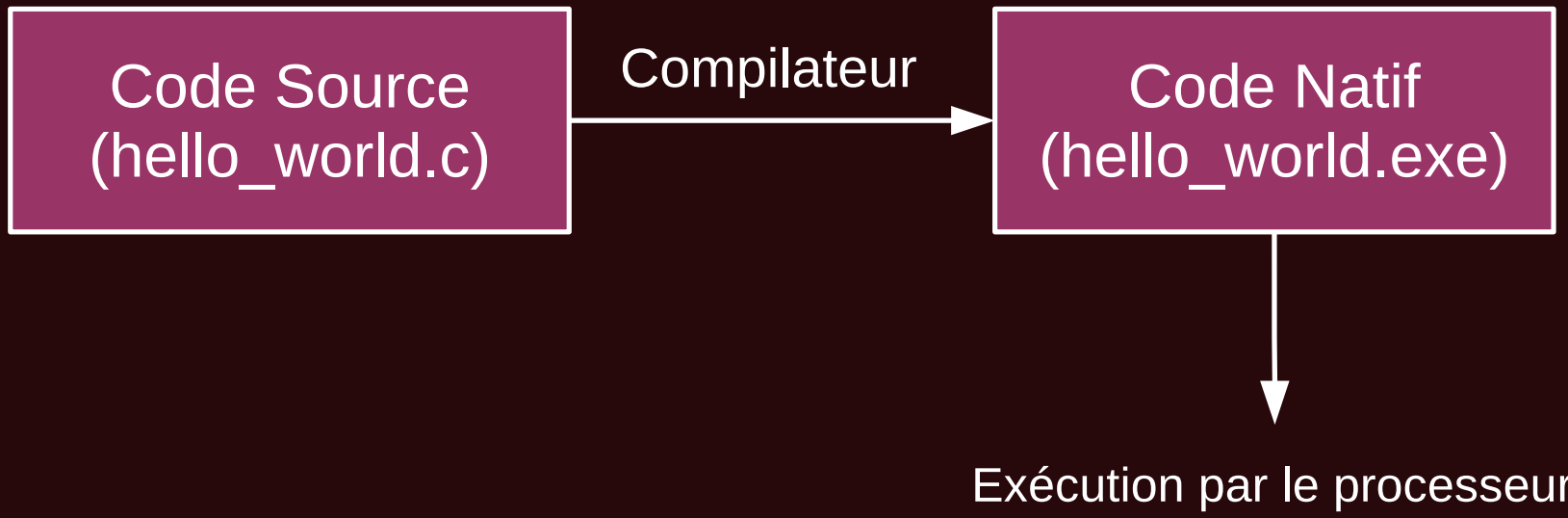
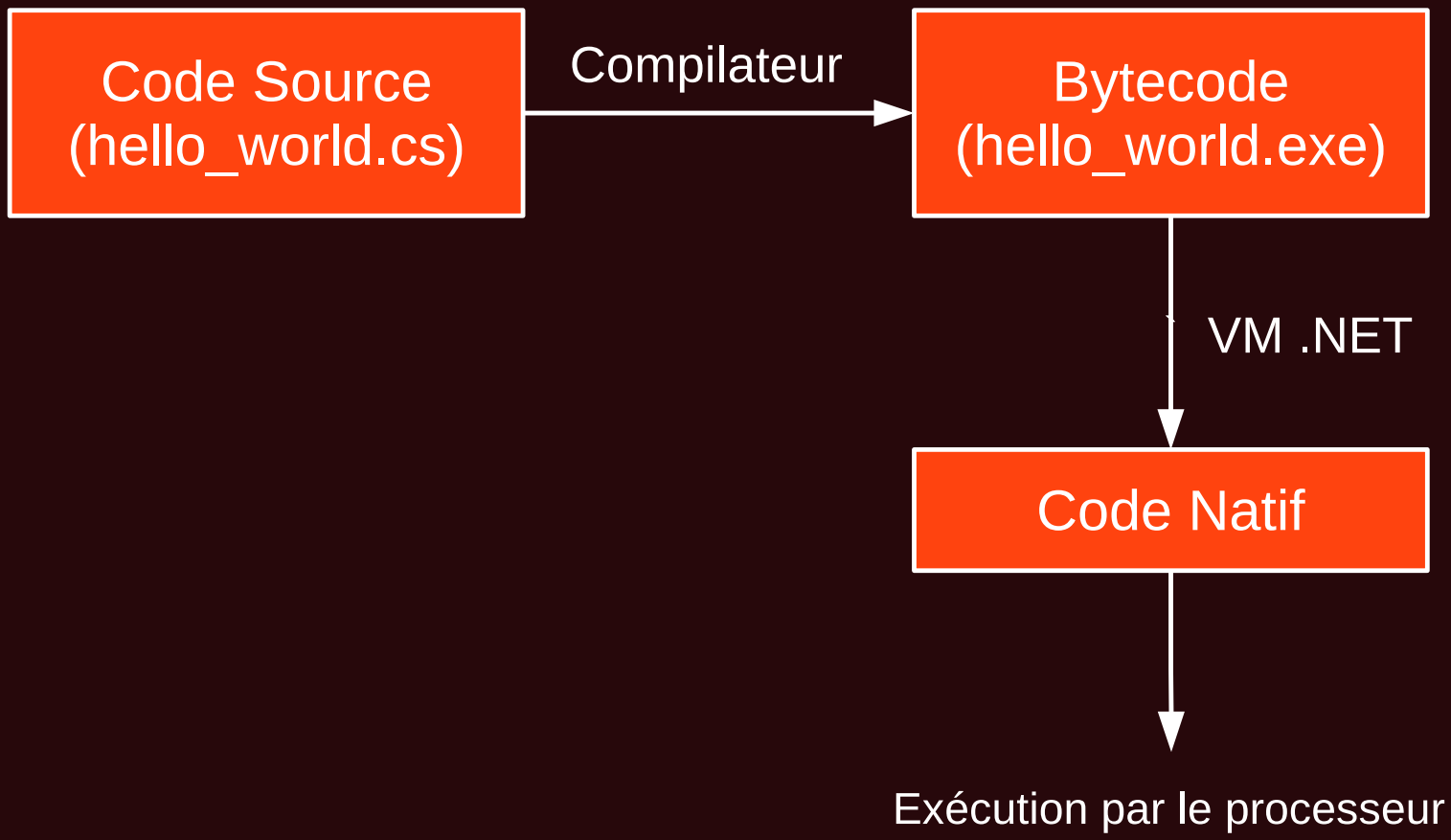
INTRODUCTION

NEODYBLUE & TOOGY

GCONFS

Menu du jour


Code Source
(hello_world.c)
Compilateur
Code Natif
(hello_world.exe)
Exécution par le processeur


Code Source
(hello_world.cs)
Compilateur
Bytecode
(hello_world.exe)
VM .NET
Code Natif
Exécution par le processeur

Programmation **impérative**

Mon ordinateur, mon esclave

- Fais moi un sandwhich ;

Mon ordinateur, mon esclave

- Fais moi un sandwhich ;
- Fais la vaisselle ;

Mon ordinateur, mon esclave

- Fais moi un sandwhich ;
- Fais la vaisselle ;
- Sors les poubelles ;

Mon ordinateur, mon esclave

- Fais moi un sandwhich ;
- Fais la vaisselle ;
- Sors les poubelles ;
- Tant que (le sol est sale) :
  { Lave le sol ; }

Mon ordinateur, mon esclave

- Fais moi un sandwhich ;
- Fais la vaisselle ;
- Sors les poubelles ;
- Tant que (le sol est sale) :
  { Lave le sol ; }
- Si (il y a du courrier) :
  { Va chercher le courrier ; }

Mon ordinateur, mon esclave

- Fais moi un sandwhich ;
- Fais la vaisselle ;
- Sors les poubelles ;
- Tant que (le sol est sale) :
  { Lave le sol ; }
- Si (il y a du courrier) :
  { Va chercher le courrier ; }

Il obéit.

Syntaxe C#

Programmation Orientée Objet *(POO)*

Field (champ)

```
public class Circle
{
    public double radius;
    public string color;
}
```

Field (champ)

```
public class Circle
{
    public double radius;
    public string color;
}
```

Method

```
public class Circle
{
    public double radius;
    public string color;

    public void setColor(string newColor)
    {
      this.color = newColor;
    }
}
```

Field (champ)

```
public class Circle
{
    public double radius;
    public string color;
}
```

Method

```
public class Circle
{
    public double radius;
    public string color;

    public void setColor(string newColor)
    {
      this.color = newColor;
    }
}
```

Property (propriété)

Snippet VS : propfull

```
public class Truc
{
    // backing field
    private int _attribut;

    public int Attribut // propriété
    {
        get { return _attribut; }
        set { _attribut = value; }
    }
}
```

Validation

```
class Thermostat
{
    private int _temperature; // backing field

    public int Temperature // propriété
    {
        get { return _temperature }
        set
        {
            if(value >= 50)
                _temperature = 50;
            else
                _temperature = value;
        }
    }
}
```

Auto-propriété

Snippet VS : prop

```
public class Objet
{
    public int Attribut { get; set; };
}
```

Auto-propriété

Snippet VS : prop

```
public class Objet
{
    public int Attribut { get; set; };
}
```

Accès privé sur le *set*

Snippet VS : propg

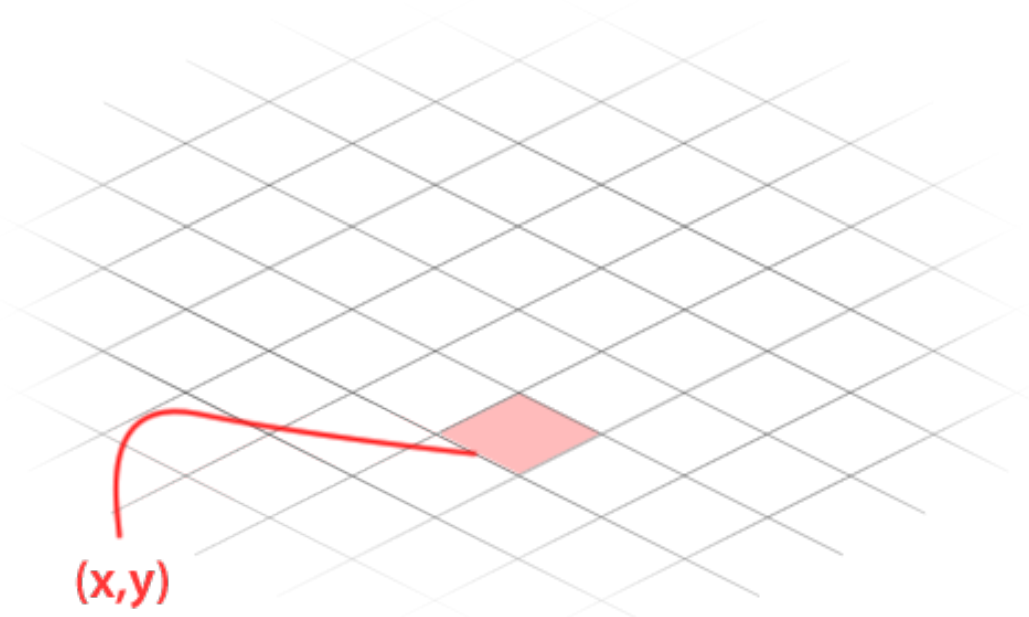
```
public class Objet
{
    public int Attribut { get; private set; };
}
```

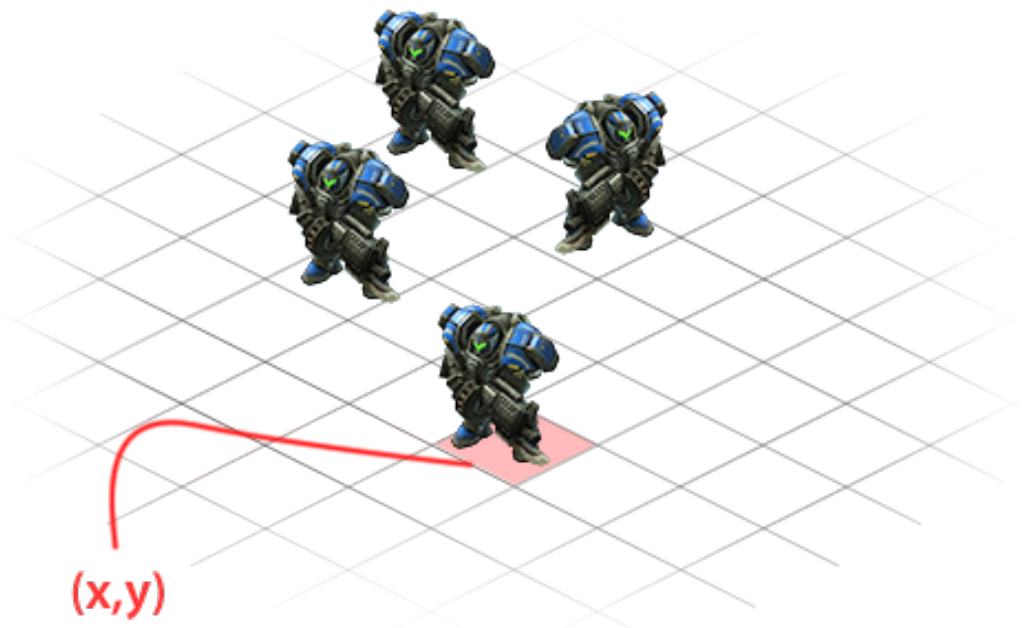
Interface

```
interface IBicycle
{
     string BrandName { get; set; }

     void ChangeSpeed(int newValue);

     void Brake();
}
```

Interface

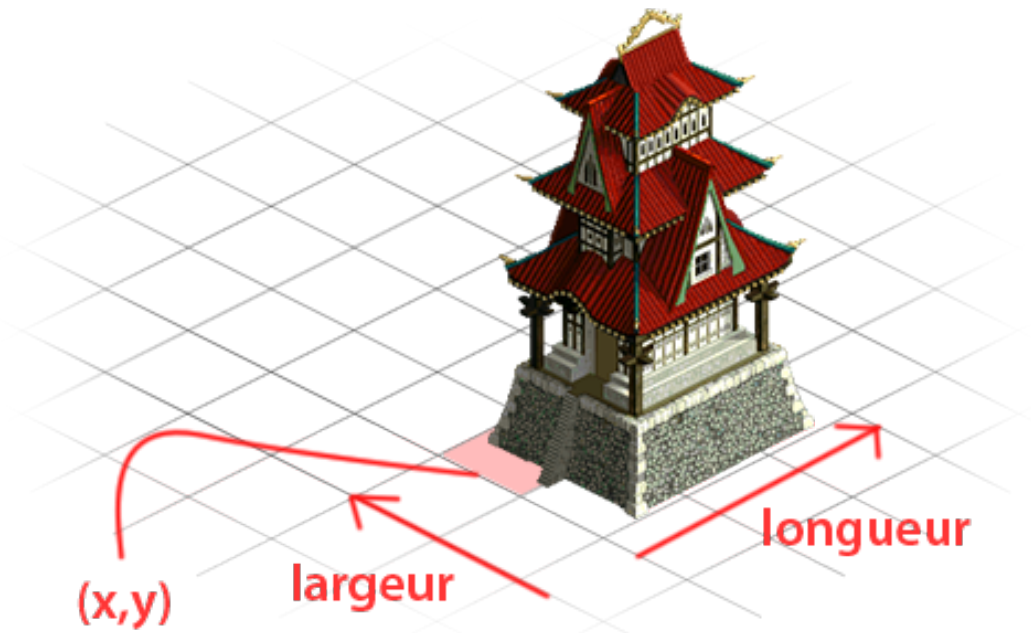
```
interface IBicycle
{
     string BrandName { get; set; }

     void ChangeSpeed(int newValue);

     void Brake();
}

class BlueBicycle  : IBicycle
{
    private string _brandName;

    public string BrandName
    {
        get { return _brandName }
        set { _brandName = lowerCase(value); }
    }
}
```


(x,y)


(x,y)

Modélisation d'une unité

```
class Unit
{
    public Tuple<int,int> Position { get; set; }

    public int HealthPoints { get; set; }
    private int _maxHealthPoints;

    public int Speed { get; private set; }

    public int Dps { get; private set; }

    public void Move(Tuple<int,int> destination)
    {
        // Bouger
    }

    public void Die()
    {
        // Mourir
    }
}
```


(x,y)
largeur
longueur

Modélisation d'un bâtiment

```
class Building
{
    public Tuple<int,int> Position { get; private set; }
    public Tuple<int,int> Size { get; private set; }

    public List<Unit> Units { get; set; }

    public int HealthPoints { get; set; }
    private int _maxHealthPoints;

    public void Die()
    {
        // Mourir
    }
}
```

```
class Unit
{
    public Tuple<int,int> Position { get; set; }

    public int HealthPoints { get; set; }
    private int _maxHealthPoints;

    public int Speed { get; private set; }

    public int Dps { get; private set; }

    public void Move(Tuple<int,int> destination)
    {
        // Bouger
    }

    public void Die()
    {
        // Mourir
    }
}
```

```
class Building
{
    public Tuple<int,int> Position { get; private set; }
    public Tuple<int,int> Size { get; private set; }

    public List<Unit> Units { get; set; }

    public int HealthPoints { get; set; }
    private int _maxHealthPoints;

    public void Die()
    {
        // Mourir
    }
}
```

```
abstract class GameItem
{
    public Tuple<int,int> Position
        { get; private set; }

    public int HealthPoints
        { get; set; }

    private int _maxHealthPoints;

    public abstract void Die();
}
```

```
abstract class GameItem
{
    public Tuple<int,int> Position
        { get; private set; }

    public int HealthPoints
        { get; set; }

    private int _maxHealthPoints;

    public abstract void Die();
}
class Building  : GameItem
{
    public Tuple<int,int> Size
        { get; private set; }

    public List<Unit> Units
        { get; set; }

    public override void Die()
    {
        // Mourir
    }
}
```

```
abstract class GameItem
{
    public Tuple<int,int> Position
        { get; private set; }

    public int HealthPoints
        { get; set; }

    private int _maxHealthPoints;

    public abstract void Die();
}

class Building  : GameItem
{
    public Tuple<int,int> Size
        { get; private set; }

    public List<Unit> Units
        { get; set; }

    public override void Die()
    {
        // Mourir
    }
}
```

```
class Unit  : GameItem
{
    public int Speed
        { get; private set; }

    public int Dps
        { get; private set; }

    public void Move(
        Tuple<int,int> destination)
    {
        // Bouger
    }

    public override void Die()
    {
        // Mourir
    }
}
```

```
abstract class GameItem
{
    public Tuple<int,int> Position
        { get; private set; }

    public int HealthPoints
        { get; set; }

    private int _maxHealthPoints;

    public abstract void Die();
}

class Building  : GameItem
{
    public Tuple<int,int> Size
        { get; private set; }

    public List<Unit> Units
        { get; set; }

    public override void Die()
    {
        // Mourir
    }
}
```

```
class Unit  : GameItem
{
    public int Speed
        { get; private set; }

    public int Dps
        { get; private set; }

    public void Move(
        Tuple<int,int> destination)
    {
        // Bouger
    }

    public override void Die()
    {
        // Mourir
    }
}
```

Héritage

```
class MyClass
{
    public void MyFonction()
    {
        List<GameItem> gameItems = new List<GameItem>();

        gameItems.Add(new Unit());
        gameItems.Add(new Building());
    }
}
```

Virtual methods

```
abstract class Unit
{
    public virtual void Die()
    {
        // A single death is a tragedy; a million deaths is a statistic.
    }
}
```

Virtual methods

```
abstract class Unit
{
    public virtual void Die()
    {
        // A single death is a tragedy; a million deaths is a statistic.
    }
}

class Bomber  : Unit
{
    public override void Die()
    {
        // Kill everyone around before actually dying

        base.Fonction(parameter);
    }
}
```

Quelques conseils

Des choses pas claires ?