

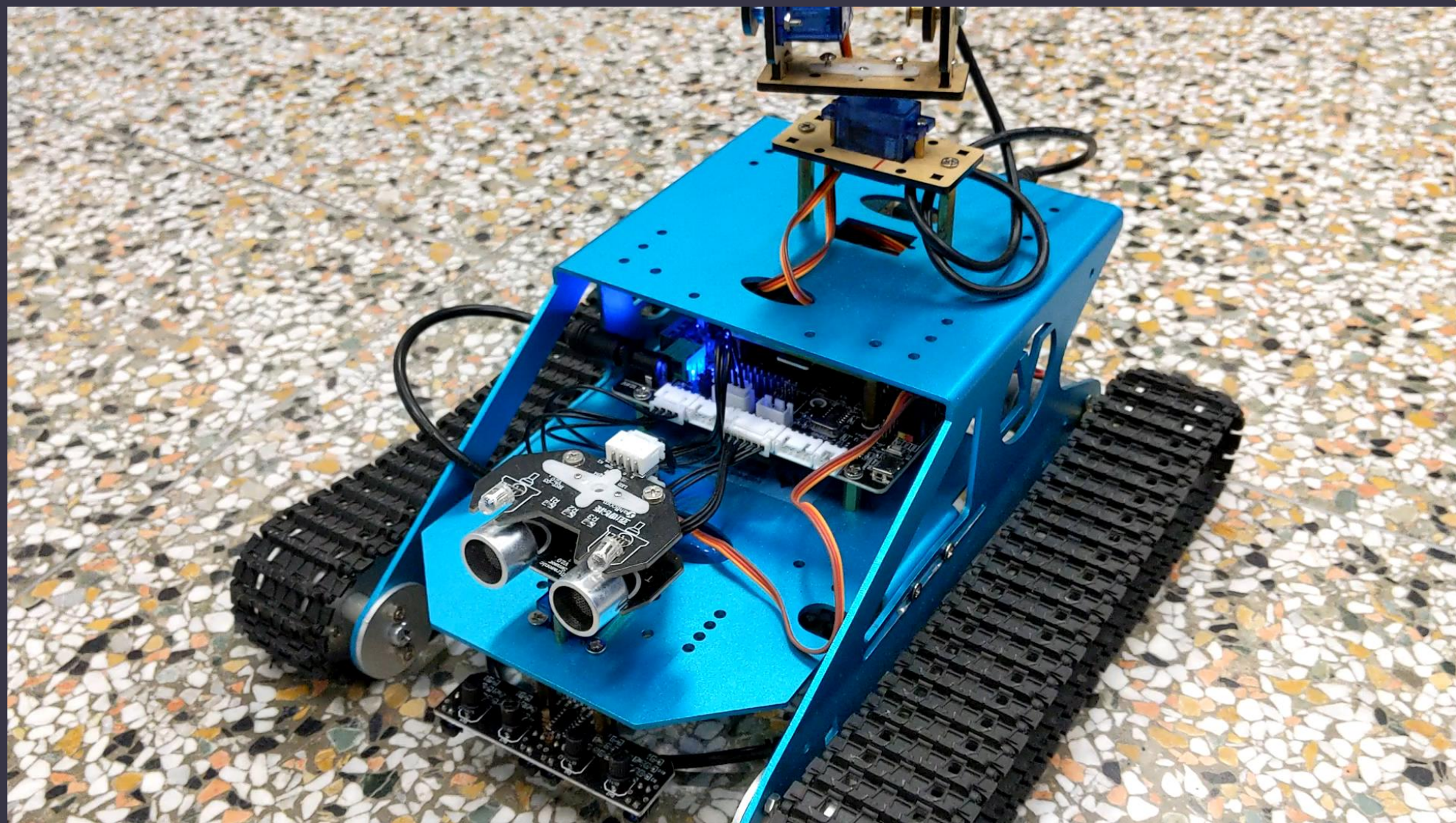
Méthodes

Les méthodes des objets

Les méthodes sont les fonctions des objets/classes. Ce sont les méthodes qui vont permettre à nos objets d'avoir des comportements des responsabilités.

C'est quoi un objet avec des responsabilités?

- Une facture qui calcule son total
- Un ennemi dans un jeu qui bouge de manière autonome
- Une réservation qui peut déterminer un conflit



Le mini-tank

- Reçoit des paquets TCP/IP
- Buzzer pour Klaxon
- LED
- Camera
- Stepper motor pour la caméra
- Moteur de chaque chenille
- Capteur Ultrason
- Mode Autopilot avec Ultrasonic
- Mode Tracking de ligne au sol

Code

Mise en situation

```
1 // reference
class CompteInvestisseur
{
    const int LimiteADecouvert = 500;
    private int fondsDisponible;
    private int fraisDus;

    0 références
    public CompteInvestisseur(int fondsInitiaux)
    {
        this.fondsDisponible = fondsInitiaux;
        fraisDus = 0;
    }

    0 références
    public int FraisDus { get => fraisDus; set => fraisDus = value; }
}
```

Exemple de méthode simple


Une méthode, c'est une fonction d'un objet. Cela s'écrit donc de la même manière.



```
public void DeposerFonds(int fondsADeposer)
{
    fondsDisponible += fondsADeposer;
}
```


Exemple de méthode simple


Si on souhaite que la méthode soit accessible hors de l'objet, par exemple dans la forme, la méthode doit être publique.



```
public void DeposerFonds(int fondsADeposer)
{
    fondsDisponible += fondsADeposer;
}
```

Exemple de méthode simple


On indique ensuite le type de retour.



```
public void DeposerFonds(int fondsADeposer)
{
    fondsDisponible += fondsADeposer;
}
```

Exemple de méthode simple

Le nom de la fonction.



```
public void DeposerFonds(int fondsADeposer)
{
    fondsDisponible += fondsADeposer;
}
```

Exemple de méthode simple


Les paramètres requis



```
public void DeposerFonds(int fondsADeposer)
{
    fondsDisponible += fondsADeposer;
}
```

Exemple de méthode simple

Ce que la fonction/méthode doit faire.



```
public void DeposerFonds(int fondsADeposer)
{
    fondsDisponible += fondsADeposer;
}
```

Mise en situation

On souhaite qu'un client ne puisse pas effectuer un retrait de plus 10000\$. On laisse une limite de 500\$ à découvert au client. Toutefois chaque retrait qui dépasse les fonds disponibles doit faire l'objet de frais, et ce même si celui-ci est refusé.

Exemple plus complexe

```
public void RetirerFonds(int fondsARetirer)
{
    if(fondsARetirer > 10000)
    {
        throw new Exception("i am afraid i can't let you do that.");
    }
    if (fondsDisponible < fondsARetirer)
    {
        fraisDus++; // :)
    }
    if (fondsDisponible + LimiteADecouvert < fondsARetirer)
    {
        throw new Exception("Fonds insuffisants");
    }
    fondsDisponible -= fondsARetirer;
}
```

Cette méthode permet à l'objet du compte d'être autonome et de faire la gestion des retraits.

Exemple plus complexe

```
public void RetirerFonds(int fondsARetirer)
{
    if(fondsARetirer > 10000)
    {
        throw new Exception("i am afraid i can't let you do that.");
    }
    if (fondsDisponible < fondsARetirer)
    {
        fraisDus++; // :)
    }
    if (fondsDisponible + LimiteADecouvert < fondsARetirer)
    {
        throw new Exception("Fonds insuffisants");
    }
    fondsDisponible -= fondsARetirer;
}
```

On empêche le retrait qui dépasse la limite à découvert.

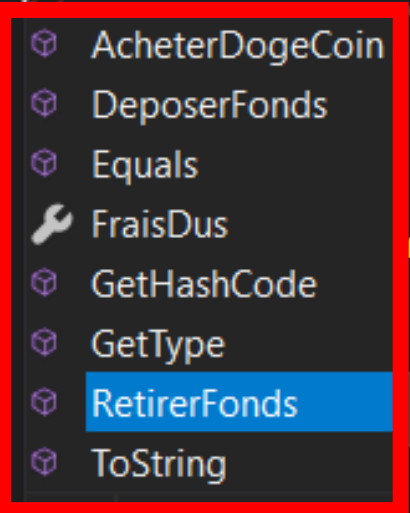
Exemple plus complexe

```
public void RetirerFonds(int fondsARetirer)
{
    if(fondsARetirer > 10000)
    {
        throw new Exception("i am afraid i can't let you do that.");
    }
    if (fondsDisponible < fondsARetirer)
    {
        fraisDus++; // :)
    }
    if (fondsDisponible + LimiteADecouvert < fondsARetirer)
    {
        throw new Exception("Fonds insuffisants");
    }
    fondsDisponible -= fondsARetirer;
}
```

Si la situation le permet, l'objet autonome va déduire les fonds.

Dans le programme

```
try
{
    compte.
}
catch
{
}
```



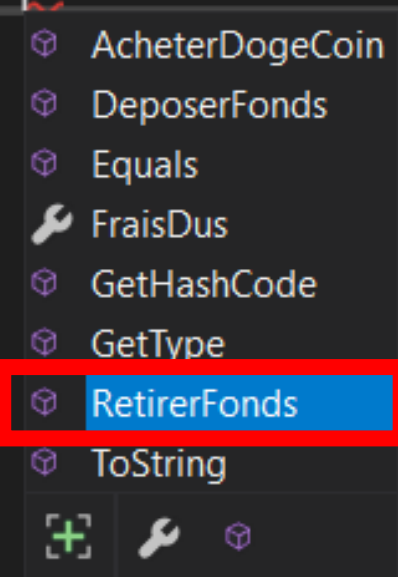
- AcheterDogeCoin
- DeposerFonds
- Equals
- FraisDus
- GetHashCode
- GetType
- RetirerFonds**
- ToString

```
void CompteInvestisseur.RetirerFonds(int fondsARetirer)
```

Les méthodes publiques sont accessibles.

Dans le programme

```
try
{
    compte.
}
catch
{
}
```

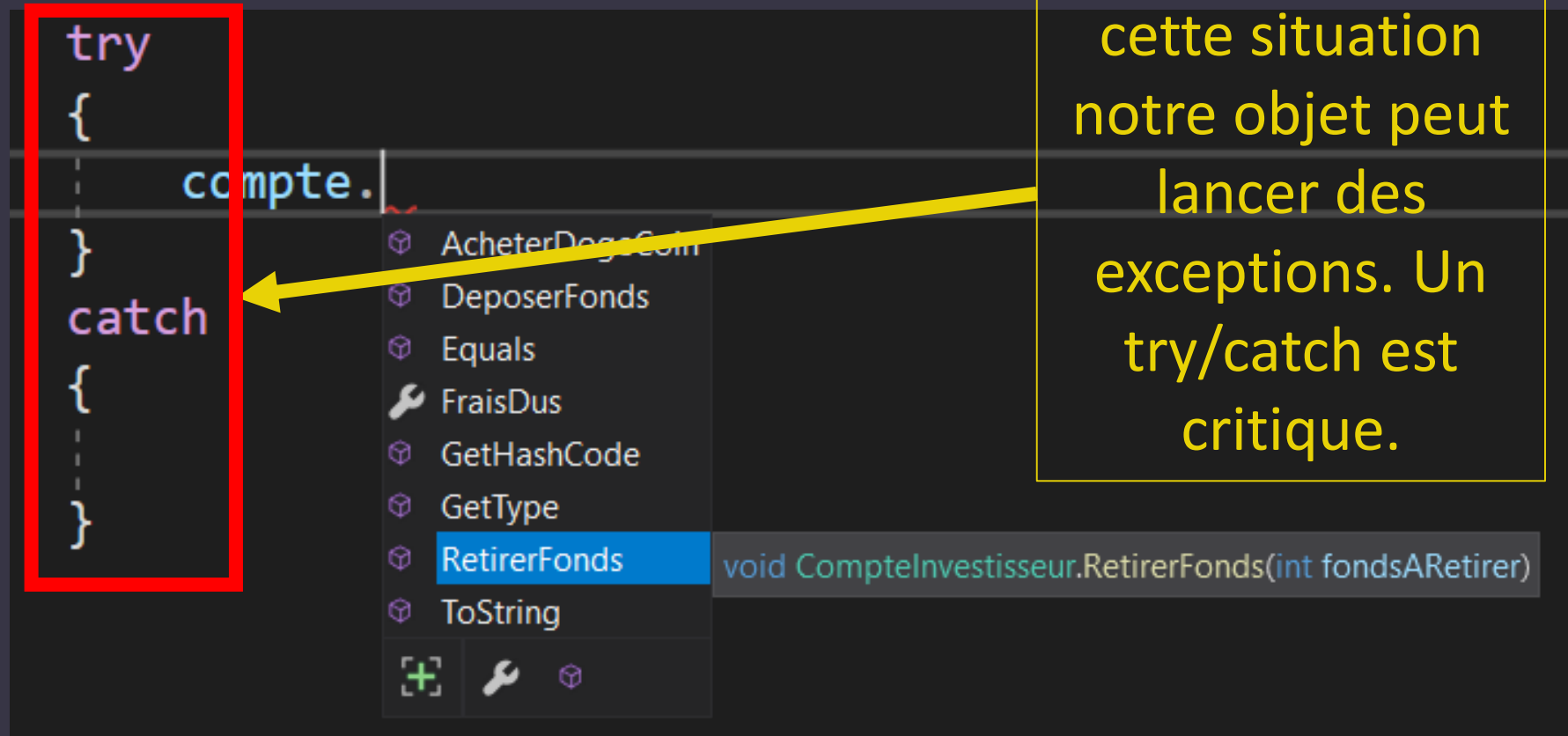


- AcheterDogeCoin
- DeposerFonds
- Equals
- FraisDus
- GetHashCode
- GetType
- RetirerFonds**
- ToString

void CompteInvestisseur.RetirerFonds(int fondsARetirer)

Les méthodes publiques sont accessibles.

Dans le programme



The screenshot shows a code editor with a `try` block and a `catch` block. The `try` block is highlighted with a red rectangle. A yellow arrow points from the `catch` block to a text box on the right. Below the `try` block, there is a list of methods: `AcheterDogsCoin`, `DeposerFonds`, `Equals`, `FraisDus`, `GetHashCode`, `GetType`, `RetirerFonds` (highlighted in blue), and `ToString`. To the right of the `RetirerFonds` method, the signature `void CompteInvestisseur.RetirerFonds(int fondsARetirer)` is visible.

```
try
{
    compte.
}
catch
{
}
```

Puisque dans cette situation notre objet peut lancer des exceptions. Un try/catch est critique.

`RetirerFonds`
`void CompteInvestisseur.RetirerFonds(int fondsARetirer)`

Autre possibilité

```
public enum ResultatOperation
{
    OK,
    FondInsuffisant,
    RetraitTropVolumineux
}
```

Au lieu d'une exception, il aurait peut-être été préférable de faire un enum avec les retours possibles.

Autre possibilité

```
public ResultatOperation RetirerFonds(int fondARetirer)
{
    if (fondARetirer > 10000)
    {
        return ResultatOperation.RetraitTropVolumineux;
    }
    if (fondsDisponible < fondARetirer)
    {
        fraisDus++; // :)
    }
    if (fondsDisponible + LimiteADecouvert < fondARetirer)
    {
        return ResultatOperation.FondInsuffisant;
    }
    fondsDisponible -= fondARetirer;
    return ResultatOperation.OK;
}
```

Retour d'une valeur qui indique le résultat de l'opération.

Code dans la forme

```
CompteInvestisseur compte = new CompteInvestisseur(500);
CompteInvestisseur.ResultatOperation resultatDuRetrait = compte.RetirerFonds(500);

switch (resultatDuRetrait)
{
    case CompteInvestisseur.ResultatOperation.OK:
        MessageBox.Show("Retrait effectué avec succès");
        break;
    case CompteInvestisseur.ResultatOperation.FondInsuffisant:
        MessageBox.Show("Fonds insuffisant");
        break;
    case CompteInvestisseur.ResultatOperation.RetraitTropVolumineux:
        MessageBox.Show("Parler avec notre gestionnaire");
        break;
    default:
        break;
}
```

Surcharge

Il est possible plusieurs méthodes avec le même nom à condition d'avoir des paramètres différents.