



Déboguer une application

Programmation de base sur PC

Plan du cours

- Les breakpoints
 - Locals
 - Watch
 - Call Stack
- Le panneau de contrôle

Les breakpoints

Locals

Watch

Call Stack

À l'aide !

- Ginette aime beaucoup ses étudiants et veut que tous et chacun puisse exploiter son plein potentiel.
- Mais là elle fait face à un problème de taille, elle ne sais pas comment aider d'avantage les étudiants lors de la création de leur code contenant des erreurs.
- Elle fait des lecture et entend parler du « débogueur »...



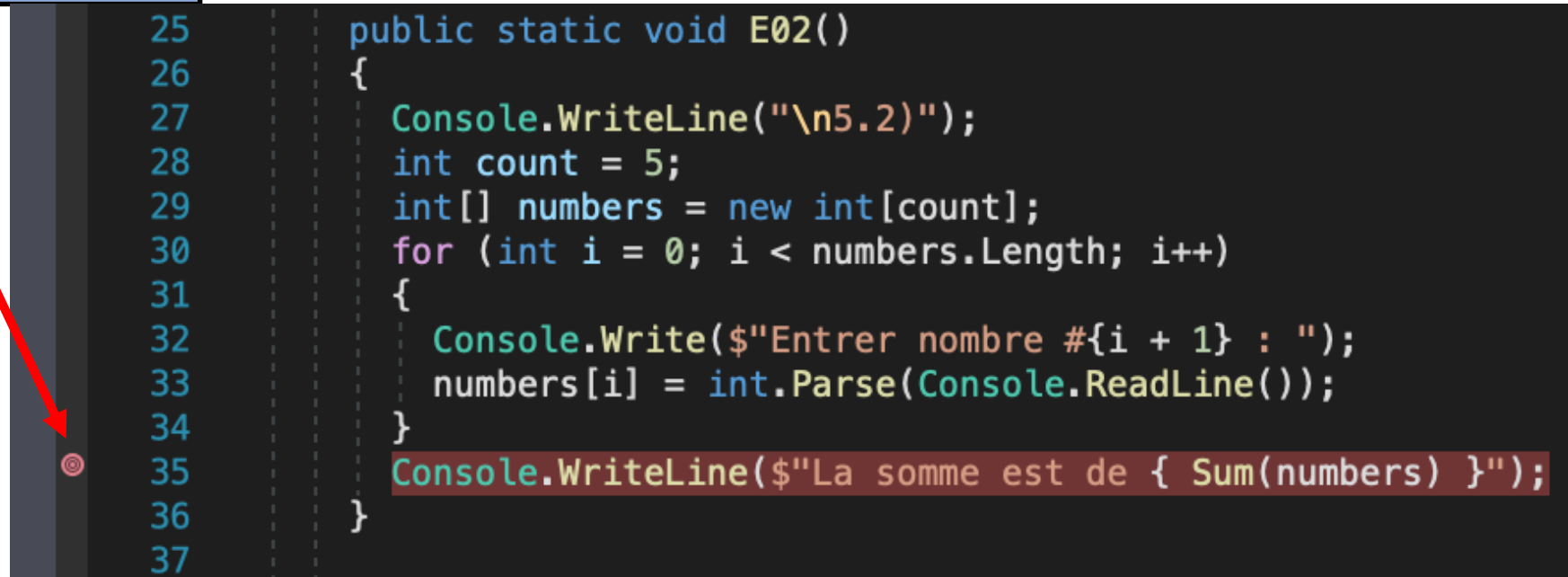
Créer un breakpoint

- Un breakpoint (point d'arrêt en français) c'est un point rouge que nous mettons sur une ligne de code pour que le logiciel fasse une pause à cet endroit lors de l'exécution du code.
- Lorsque le logiciel sera sur pause, il sera entre autres possible de connaître plusieurs éléments du contexte en cours :
 1. Locals: Liste des variables accessibles lors de la pause et leur valeurs.
 2. Watch: Valeur d'un élément précis que l'on a décidé d'observer.
 3. Call Stack: Liste ordonnée des appels de fonctions.

Créer un breakpoint

- Cliquez sur la bande des breakpoint à une ligne précise pour en créer un.

Voici un point d'arrêt qui fera en sorte que le logiciel se mette en pause à la ligne #35



The screenshot shows a code editor with a dark background. On the left, a vertical bar contains line numbers 25 through 37. A red circle with a dot in the center, representing a breakpoint, is positioned next to line 35. A red arrow points from a text box above to this breakpoint. The code on the right is in C# and defines a static method E02(). The code is as follows:

```
public static void E02()
{
    Console.WriteLine("\n5.2");
    int count = 5;
    int[] numbers = new int[count];
    for (int i = 0; i < numbers.Length; i++)
    {
        Console.Write($"Entrer nombre #{i + 1} : ");
        numbers[i] = int.Parse(Console.ReadLine());
    }
    Console.WriteLine($"La somme est de { Sum(numbers) }");
}
```

Locals: Inspecter la valeur des variables

- Ouvrez l'onglet « Locals » pour obtenir la valeur des variables accessibles

On peut ainsi observer le tableau « numbers » prendre ses valeurs une à une après l'entrée au clavier

Le type de chacune des variable disponibles en mémoire actuellement est également indiqué

```
5.2)
Entrer nombre #1 : 90
Entrer nombre #2 : 70
Entrer nombre #3 : 
```

Call Stack

Locals

Watch

Threads

Terminal - Notes

Name	Value	Type
count	5	int
▼ numbers	{int[5]}	int[]
[0]	90	int
[1]	70	int
[2]	0	int
[3]	0	int
[4]	0	int
i	2	int

Watch: Observer la valeur d'une expression

- Ouvrez l'onglet « Watch » pour obtenir la valeur d'une expression

Faite appel à la fonction Sum() et watch vous retournera le résultat selon les nombres entrés au moment du breakpoint

Toute les expressions valides sont acceptées, même si elle n'ont aucun rapport avec le code

```
5.2)
Entrer nombre #1 : 90
Entrer nombre #2 : 70
Entrer nombre #3 : 
```

Call Stack			Locals			Watch		
						Name	Value	Type
						Sum(numbers)	160	int
						Math.Round(10.562234,2)	10.56	double
						+ Add item to watch		

Appuyez sur « Add item to watch » pour ajouter n'importe quel type d'expression à ajouter

Call stack: Voir l'ordre des appels de fonctions

- Ouvrez l'onglet « Call Stack » pour visualiser la pile d'appel

1

```
static void Main(string[] args)
{
    Exercices05.Run();
}
```

2

```
public static void Run()
{
    E05();
}
```

3

```
public static int Sum(int[] numbers)
{
    int sum = 0;
    for (int i = 0; i < numbers.Length; i++)
    {
        sum += numbers[i];
    }
    return sum;
}
```

	Call Stack	Locals	Watch	Threads	Terminal - Notes
	Name	File			
3-	→ Projet01.Exercices05.E02() Line 33()	/Users/guilmarc/Repositories/Shawinigan/DevLo/Projet01/Notes/Exercices05.cs:33			
2-	Projet01.Exercices05.Run() Line 9(Arguments)	/Users/guilmarc/Repositories/Shawinigan/DevLo/Projet01/Notes/Exercices05.cs:9			
1-	Projet01.Program.Main(string[] args) Line 14(Arguments)	/Users/guilmarc/Repositories/Shawinigan/DevLo/Projet01/Notes/Program.cs:14			
	[External Code]				

À l'aide !

- Ginette est fébrile car elle viens de réaliser à quel point les breakpoints seront utilise à sa compréhension du code et des appels de fonctions.
- Elle aimerai maintenant savoir si elle peut demander à ses élèves d'utiliser cela dans la classe mais elle a encore un questionnaire: « Comment se déplacent-on dans le code par la suite pour continuer l'exécution ? ».
- Bonne question Ginette 😊





Le panneau de contrôle

Le panneau de contrôle

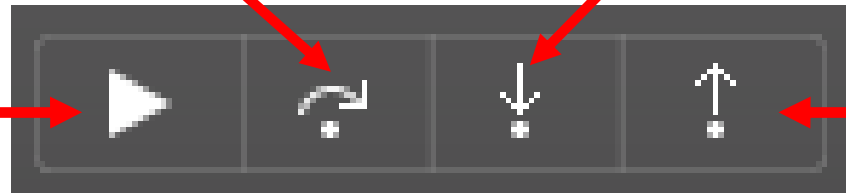
- Vous aurez besoin de ces quatre boutons pour vous déplacer dans le code

Continue execution : Relancer le code jusqu'au prochain breakpoint d'exécution

Step Over : Prochaine ligne de code sans entrer dans les fonctions rencontrées

Step Into : Prochaine ligne de code en entrant dans les fonctions rencontrées

Step Out : Faire un pas en arrière dans la pile d'appel des fonctions. Sortie de l'appel de la fonction en cours



The background of the image is a dark blue gradient filled with a pattern of binary code (0s and 1s) in a lighter blue color. The binary code is arranged in a way that creates a sense of depth and movement, with some digits appearing larger and more prominent than others. In the center of the image, there is a white rectangular box with a thin black border. Inside this box, the text "La pratique" is written in a clean, white, sans-serif font.

La pratique