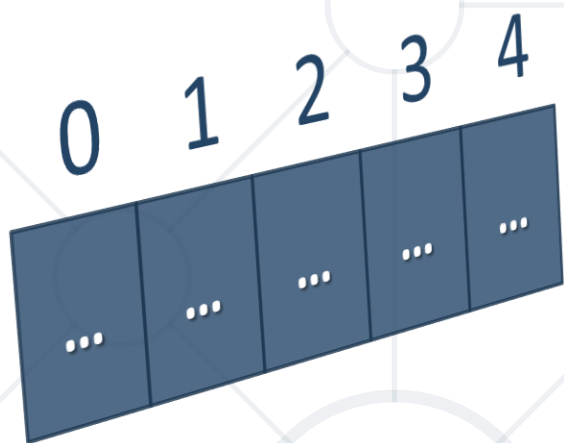


Stacks and Queues

Processing Sequences of Elements



SoftUni Team

Technical Trainers



SoftUni



Software University

<https://about.softuni.bg/>

sli.do

#csharp-advanced

1. The "**Stack**" Data Structure (**LIFO** - last in, first out)

- The Class **Stack<T>**
- Push(), Pop(), Peek(),
ToArray(), Contains() and Count

2. The "**Queue**" Data Structure (**FIFO** - first in, first out)

- The Class **Queue<T>**
- Enqueue(), Dequeue(), Peek(),
ToArray(), Contains() and Count





The "Stack" Data Structure

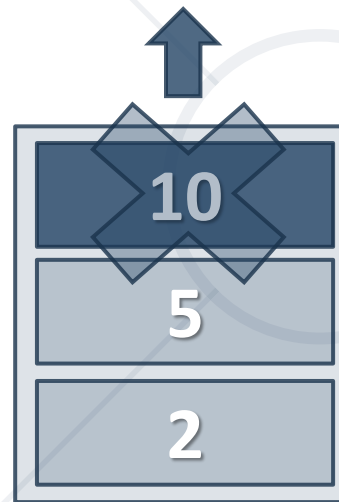
Using the `Stack<T>` Class

Stack – Abstract Data Type

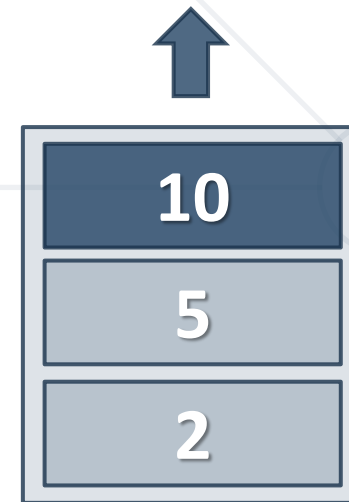
- **Stack** implements a **LIFO** (last in, first out) collection
 - **Push**: insert an element at the top of the stack
 - **Pop**: take the element from the top of the stack
 - **Peek**: retrieve the topmost element without removing it



Push



Pop



Peek

Push() – Adds an Element On Top of the Stack

5

Stack<int>

Count:

3

Pop() – Returns and Removes the Last Element

Stack<int>

2

10

5

Count:

2

Peek() - Returns the Last Element



The diagram illustrates a stack data structure. It features a central circle containing a light blue rectangular box with the text `Stack<int>`. Below this circle is a dark blue rectangular box containing the number 5. To the right of the central circle is the text **Count:** followed by a dark blue rectangular box containing the number 1. The background is a light gray grid with several circles of varying sizes connected by thin gray lines, creating a network-like pattern.

`Stack<int>`

Count: 1

5

Problem: Reverse a String

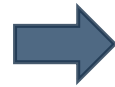
- Create a program that:
 - Reads an input string
 - **Reverses** its letters backwards using a **stack**

I Love C#



#C evoL I

Stacks and Queues



seueuQ dna skcatS

Check your solution here: <https://judge.softuni.org/Contests/Practice/Index/1445#0>

Solution: Reverse Strings

```
var input = Console.ReadLine();  
var stack = new Stack<char>();  
foreach (var ch in input)  
{  
    stack.Push(ch);  
}  
while (stack.Count > 0)  
{  
    Console.Write(stack.Pop());  
}  
Console.WriteLine();
```

Stack – Utility Methods

```
Stack<int> stack = new Stack<int>();
```

```
int count = stack.Count;
```

```
bool exists = stack.Contains(2);
```

```
int[] array = stack.ToArray();
```

```
stack.Clear();
```

```
stack.TrimExcess();
```

Retains the order
of elements

Remove all
elements

Shrink the
internal array

Problem: Stack Sum

- You are given a **list of numbers**. Push them into a stack and execute a sequence of **commands**:
 - **Add <n1> <n2>**: adds given two numbers to the stack
 - **Remove <count>**: if elements are enough, removes *count* elements
 - **End**: print the sum in the remaining elements from the stack and exit

```
1 2 3 4
ADD 5 6
REmove 3
eNd
```



Sum: 6

```
3 5 8 4 1 9
add 19 32
remove 10
add 89 22
end
```



Sum: 192

Check your solution here: <https://judge.softuni.org/Contests/Practice/Index/1445#1>

```
var input =  
    Console.ReadLine().Split().Select(int.Parse).ToArray();  
Stack<int> stack = new Stack<int>(input);  
var commandInfo = Console.ReadLine().ToLower();  
  
while (commandInfo != "end")  
{  
    var tokens = commandInfo.Split();  
    var command = tokens[0].ToLower();  
    if (command == "add")  
        // TODO: Parse the numbers and push them to the stack
```

```
else if (command == "remove") {  
    var countOfRemovedNums = int.Parse(tokens[1]);  
    if (countOfRemovedNums <= stack.Count)  
        for (int i = 0; i < countOfRemovedNums; i++) {  
            stack.Pop();  
        }  
    commandInfo = Console.ReadLine().ToLower();  
}  
var sum = stack.Sum();  
Console.WriteLine($"Sum: {sum}");
```

Problem: Matching Brackets

- We are **given an arithmetic expression** with brackets (**nesting is allowed**)
- **Extract all sub-expressions** in brackets

1 + (2 - (2 + 3) * 4 / (3 + 1)) * 5



(2 + 3)

(3 + 1)

(2 - (2 + 3) * 4 / (3 + 1))

Check your solution here: <https://judge.softuni.org/Contests/Practice/Index/1445#3>

Solution: Matching Brackets

```
var input = Console.ReadLine();
var stack = new Stack<int>();
for (int i = 0; i < input.Length; i++) {
    char ch = input[i];
    if (ch == '(') {
        stack.Push(i);
    } else if (ch == ')') {
        int startIndex = stack.Pop();
        string contents = input.Substring(
            startIndex, i - startIndex + 1);
        Console.WriteLine(contents);
    }
}
```




The "Queue" Data Structure

Using the `Queue<T>` Class

Queue – Abstract Data Type

- Queue implements a **FIFO** (first in, first out) collection

- **Enqueue:** append an element at the end of the queue



- **Dequeue:** remove the first element from the queue



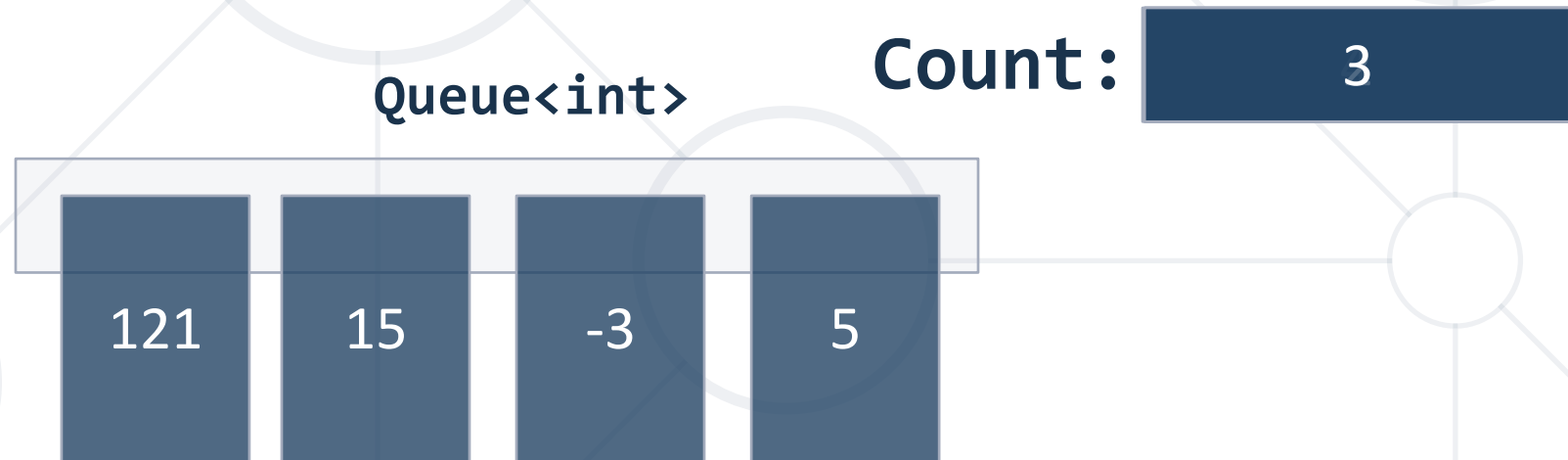
- **Peek:** retrieve the first element of the queue without removing it



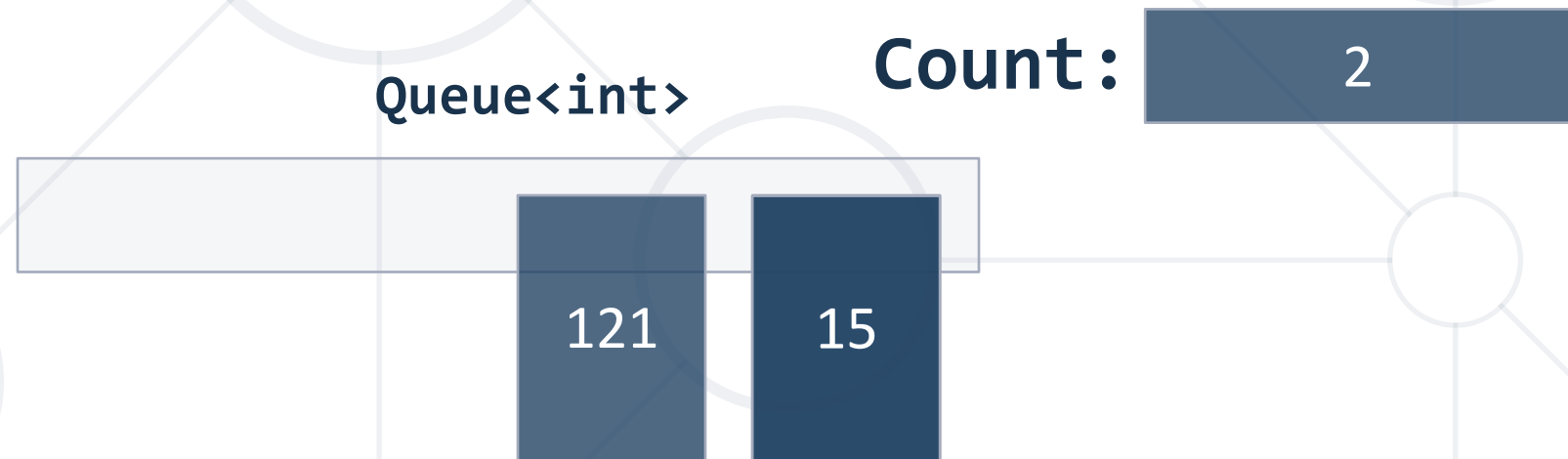
Enqueue() – Adds an Element to the Front



Deque() – Returns and Removes the First Element



Peek() – Returns the First Element



Queue – Utility Methods

```
Queue<int> queue = new  
Queue<int>();  
int count = queue.Count;  
bool exists = queue.Contains(2);  
int[] array = queue.ToArray();  
queue.Clear();  
queue.TrimExcess();
```

Remove all
elements

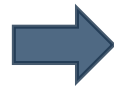
Retains the order
of elements

Resize the
internal array

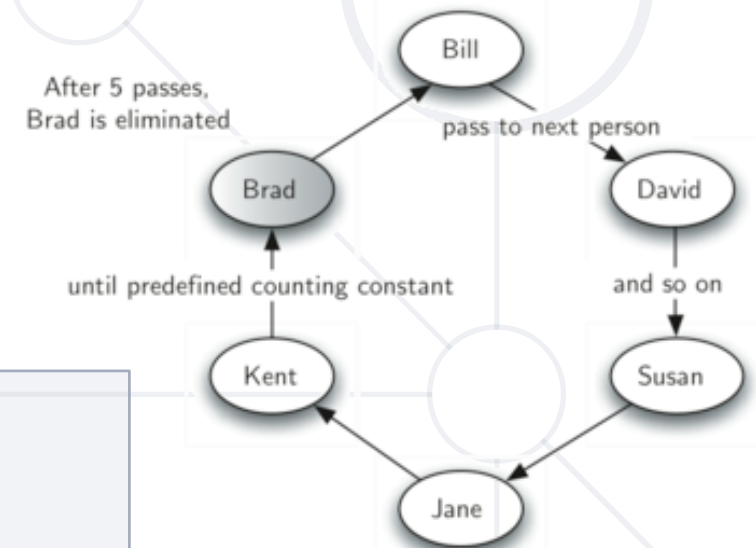
Problem: Hot Potato

- Children **form a circle** and pass a hot potato **clockwise**
- Every n^{th} toss **a child is removed** until **only one remains**
 - Upon removal** the potato is passed **along**
- Print the child that remains last

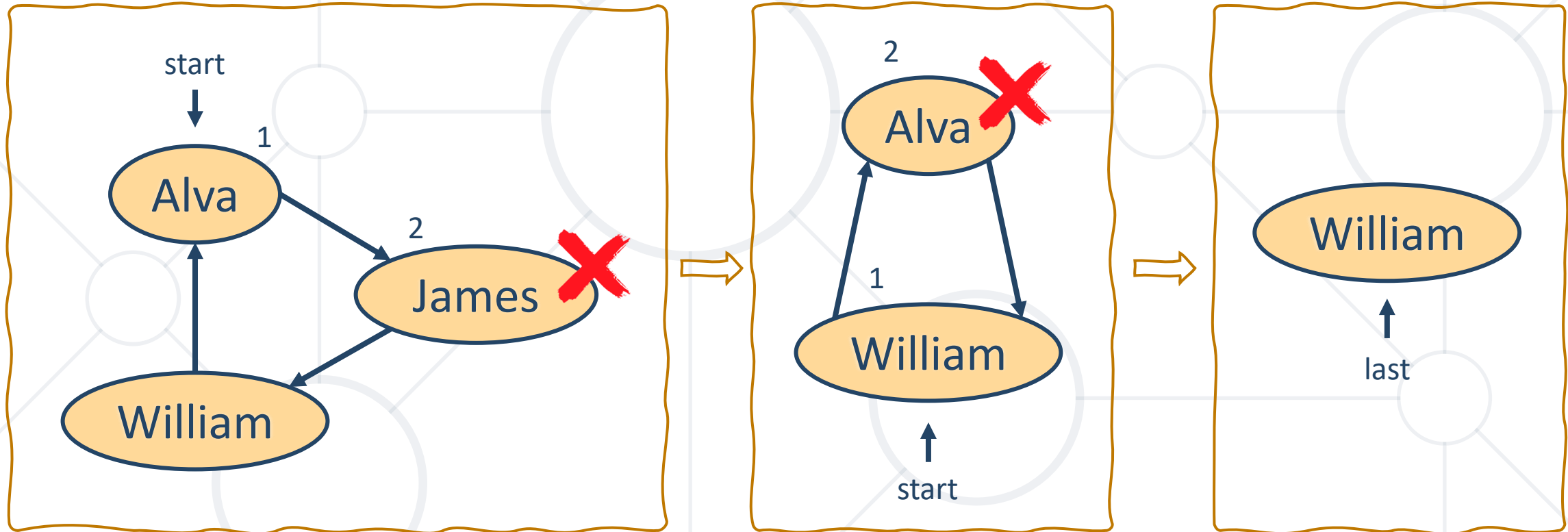
Alva James William
2



Removed James
Removed Alva
Last is William



Hot Potato: Illustration



Solution: Hot Potato

```
var children = Console.ReadLine().Split(' ');
var number = int.Parse(Console.ReadLine());
Queue<string> queue = new Queue<string>(children);
while (queue.Count > 1) {
    for (int i = 1; i < number; i++) {
        queue.Enqueue(queue.Dequeue());
    }
    Console.WriteLine($"Removed {queue.Dequeue()}");
}
Console.WriteLine($"Last in {queue.Dequeue()}");
```

Copies elements from the specified collection and keeps their order

Problem: Traffic Jam

- Cars are **queuing up** at a **traffic light**
- At every **green light**, **n** cars **pass** the crossroad
- After the **end command**, print **how many cars** have **passed**

```
3
Enzo's car
Jade's car
Mercedes CLS
Audi
green
BMW X5
green
end
```



```
Enzo's car passed!
Jade's car passed!
Mercedes CLS passed!
Audi passed!
BMW X5 passed!
5 cars passed the crossroads.
```

Solution: Traffic Jam

```
int n = int.Parse(Console.ReadLine());
var queue = new Queue<string>();
int count = 0;
string command;
while ((command = Console.ReadLine()) != "end")
{
    if (command == "green")
        // TODO: Add green light logic
    else
        queue.Enqueue(command);
}
Console.WriteLine($"{count} cars passed the crossroads.");
```

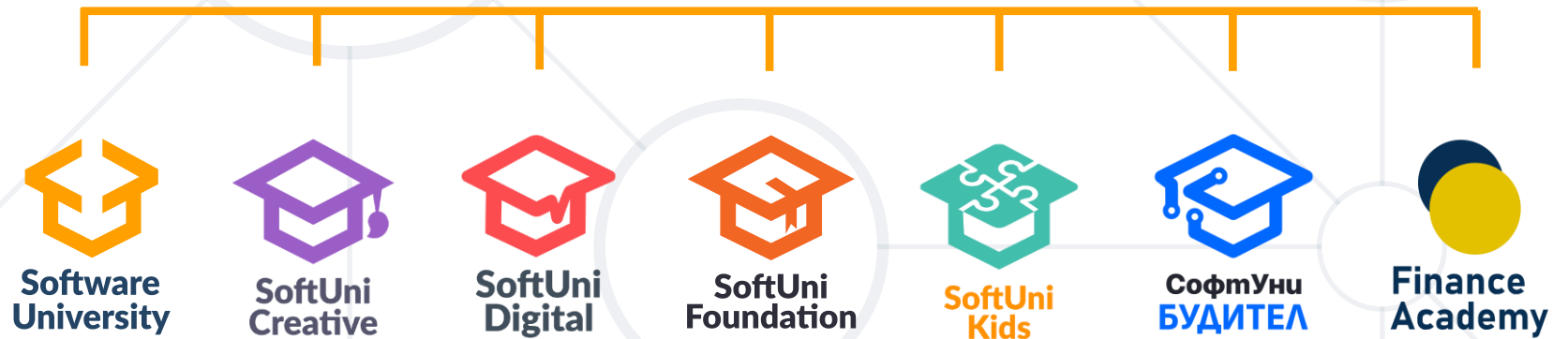
- **Stack<T>**
 - **LIFO** data structure (last-in, first-out)
 - **Push()**, **Pop()**, **Peek()**
- **Queue<T>**
 - **FIFO** data structure (first-in, first-out)
 - **Enqueue()**, **Dequeue()**, **Peek()**



Questions?



SoftUni



SoftUni Diamond Partners



THE CROWN IS YOURS



- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg, about.softuni.bg

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>

