

# Stack and Queue



**SoftUni Team**  
**Technical Trainers**



**SoftUni**



**Software University**

<https://softuni.bg>

[sli.do](https://sli.do)

**#java-advanced**

1. Stack - Last In First Out (LIFO)
  - Stack Functionality
  - Java Stack Implementation
  - Overview of All Operations
2. Queue - First In First Out(FIFO)
  - Queue Functionality
  - Queue Implementation
  - Overview of All Operations
3. Priority Queue

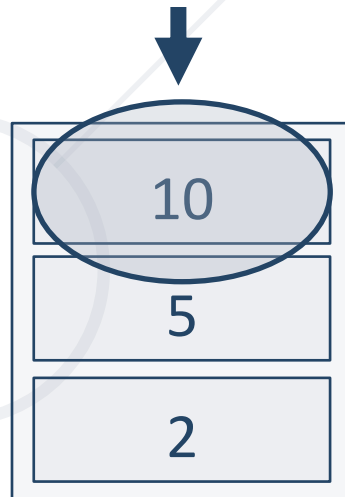




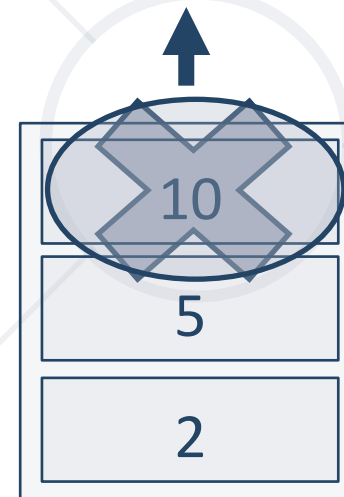
**Stack**

# Stack Functionality

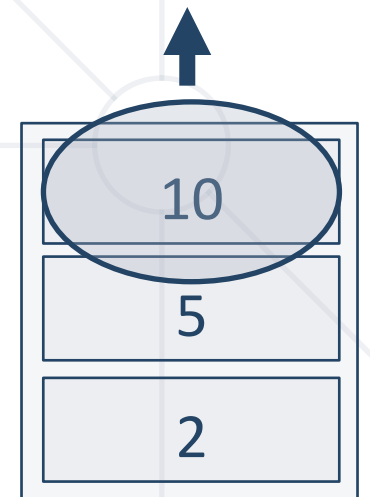
- **Stacks** provide the following functionality:
  - Pushing an element at the top of the stack
  - Popping element from the top of the stack
  - Getting the topmost element without removing it



Push



Pop



Peek

# ArrayDeque<E> – Java Stack Implementation

- Creating a Stack

```
ArrayDeque<Integer> stack = new ArrayDeque<>();
```

- Adding elements at the top of the stack

```
stack.push(element);
```

- Removing elements

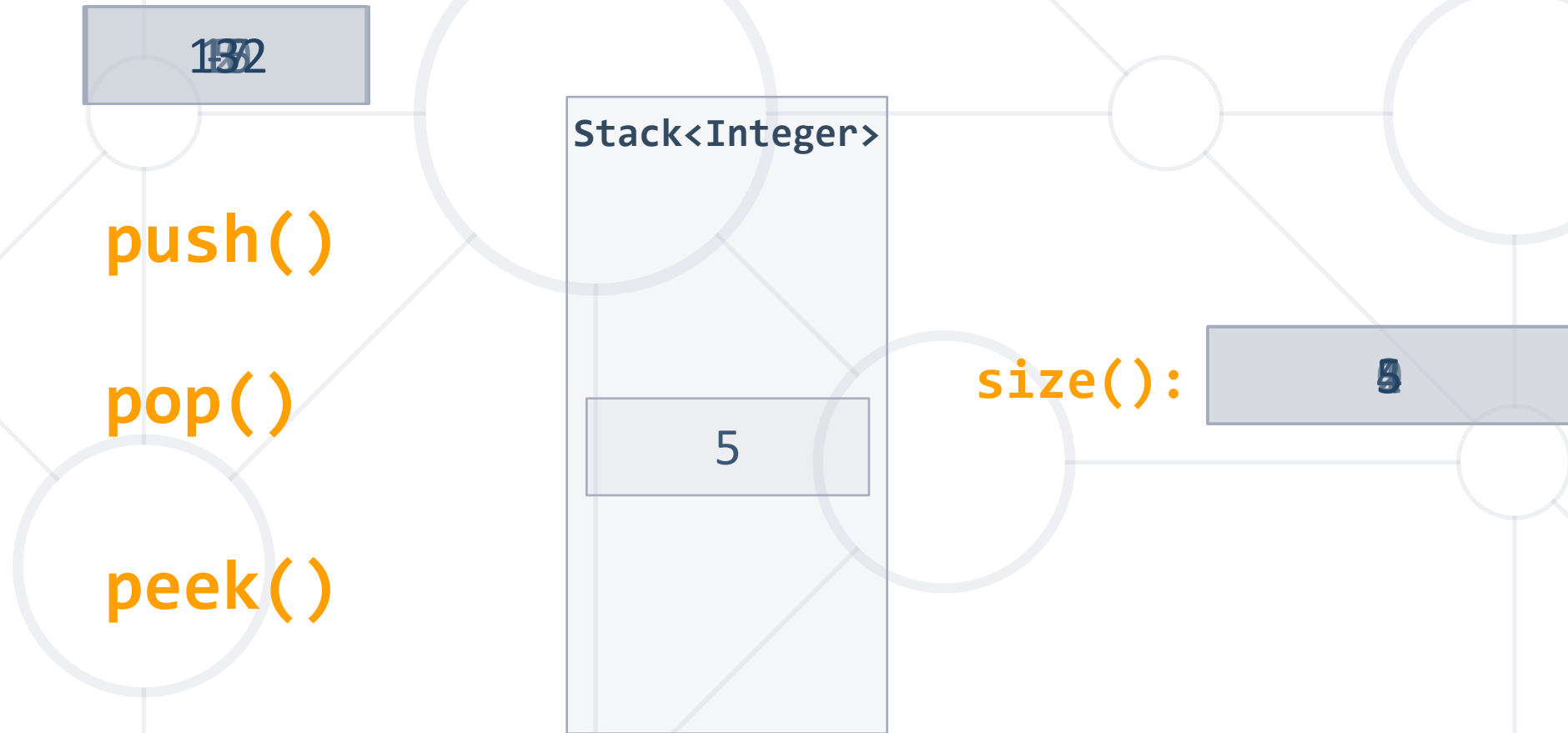
```
Integer element = stack.pop();
```

- Getting the value of the topmost element

```
Integer element = stack.peek();
```

```
ArrayDeque<Integer> stack = new ArrayDeque<>();  
  
int size = stack.size();  
boolean isEmpty = stack.isEmpty();  
boolean exists = stack.contains(2);
```

# Stack – Overview of All Operations





# Problem: Browser History

- Write a program which takes 2 types of **browser instructions**:
  - Normal navigation: a **URL** is set, given by a string
  - The string "**back**" that sets the current URL to the last set URL

| Input   |
|---|
| <code>https://softuni.bg/<br/>back<br/>https://softuni.bg/trainings/courses<br/>back<br/>https://softuni.bg/trainings/2056<br/>back<br/>https://softuni.bg/trainings/live<br/>https://softuni.bg/trainings/live/details<br/>Home</code> |



| Output   |
|--|
| <code>https://softuni.bg/<br/>no previous URLs<br/>https://softuni.bg/trainings/courses<br/>https://softuni.bg/<br/>https://softuni.bg/trainings/2056<br/>https://softuni.bg/<br/>https://softuni.bg/trainings/live<br/>https://softuni.bg/trainings/live/details</code> |

# Solution: Browser History (1)

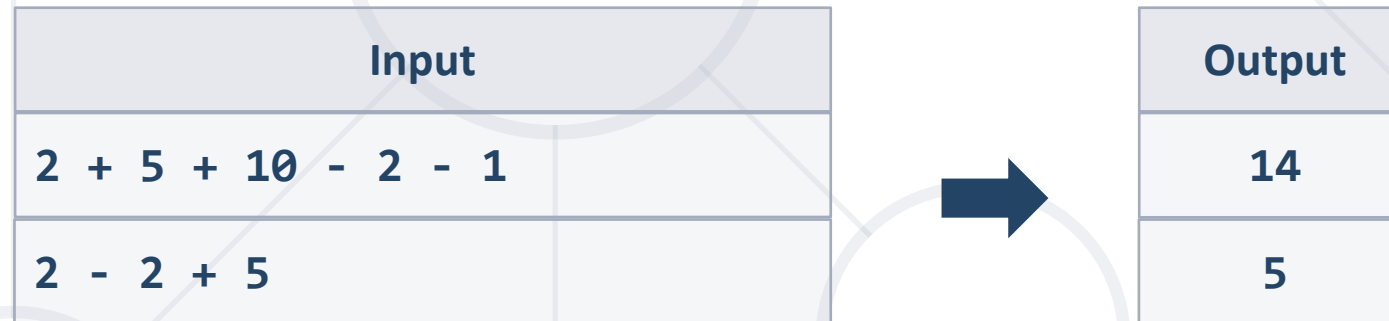
```
Scanner scanner = new Scanner(System.in);  
  
ArrayDeque<String> browser = new ArrayDeque<>();  
String line = scanner.nextLine();  
  
String current = "";  
  
// continue...
```

# Solution: Browser History (2)

```
while(!line.equals("Home")) {  
    if(line.equals("back")) {  
        if(!browser.isEmpty()) { current = browser.pop();  
        } else {  
            System.out.println("no previous URLs");  
            line = scanner.nextLine();  
            continue; }  
    } else {  
        if(!current.equals("")) { browser.push(current); }  
        current = line; }  
    System.out.println(current);  
    line = scanner.nextLine(); }
```

# Problem: Simple Calculator

- Implement a simple calculator that can evaluate simple expressions (only addition and subtraction)



# Solution: Simple Calculator (1)

```
Scanner scanner = new Scanner(System.in);  
String[] tokens = scanner.nextLine().split("\\s+");
```

Split by regex

```
Deque<String> stack = new ArrayDeque<>();  
Collections.addAll(stack, tokens);
```

Adds a collection to  
another collection

```
// continues...
```

# Solution: Simple Calculator (2)

```
while (stack.size() > 1) {  
    int first = Integer.valueOf(stack.pop());  
    String op = stack.pop();  
    int second = Integer.valueOf(stack.pop());  
  
    switch (op)  
    {  
        case "+": stack.push(String.valueOf(first + second));  
        break;  
        case "-": stack.push(String.valueOf(first - second));  
        break;  
    }  
  
    System.out.println(stack.pop());  
}
```

# Problem: Decimal to Binary Converter

- Create a converter which takes a **decimal number** and **converts it into a binary number**

| Input |
|-------|
| 10    |
| 1024  |



| Output     |
|------------|
| 1010       |
| 1000000000 |

# Solution: Decimal to Binary Converter

```
Scanner scanner = new Scanner(System.in);
int decimal = Integer.valueOf(scanner.nextLine());

ArrayDeque<Integer> stack = new ArrayDeque<>();

// TODO: check if number is 0

while (decimal != 0)
    stack.push(decimal % 2);
    decimal /= 2;

while (!stack.isEmpty())
    System.out.print(stack.pop());
```



# Problem: Matching Brackets

- We are given an arithmetical expression with brackets (with nesting)
- Goal: extract all sub-expressions in brackets

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



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

# Solution: Matching Brackets (1)

```
Scanner scanner = new Scanner(System.in);  
String expression = scanner.nextLine();  
  
Deque<Integer> stack = new ArrayDeque<>();  
  
// continue...
```

# Solution: Matching Brackets (2)

```
for (int i = 0; i < expression.length(); i++) {  
    char ch = expression.charAt(i);  
    if (ch == '(')  
        stack.push(i);  
    else if (ch == ')')  
        int startIndex = stack.pop();  
        String contents =  
            expression.substring(startIndex, i + 1);  
        System.out.println(contents);  
}
```



**Queues**

- **First In First Out**



# Queue – Abstract Data Type

- Queues provide the following functionality:

- Adding an element at the end of the queue



- Removing the first element from the queue



- Getting the first element of the queue without removing it



- Creating a Queue

```
ArrayDeque<Integer> queue = new ArrayDeque<>();
```

- Adding elements at the end of the queue

```
queue.add(element);  
queue.offer(element);
```

- **add()** – throws exception if queue is full
- **offer()** – returns false if a queue is full

- Removing elements

```
element = queue.remove();  
element = queue.poll();
```

- **remove()** - throws exception if queue is empty
- **poll()** - returns null if queue is empty

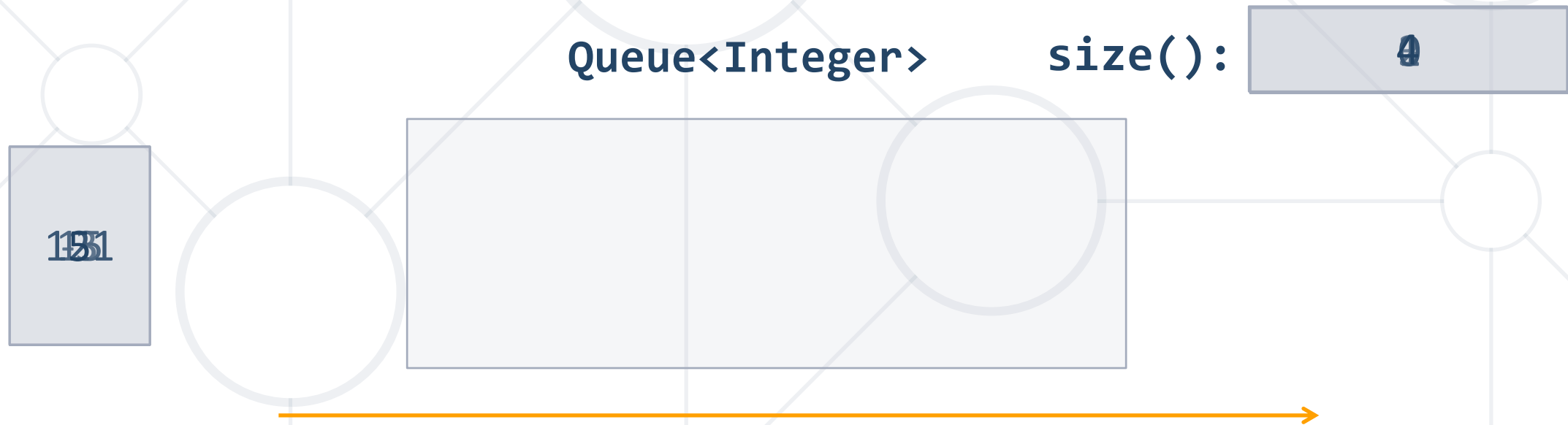
- Check first element

```
element = queue.peek();
```



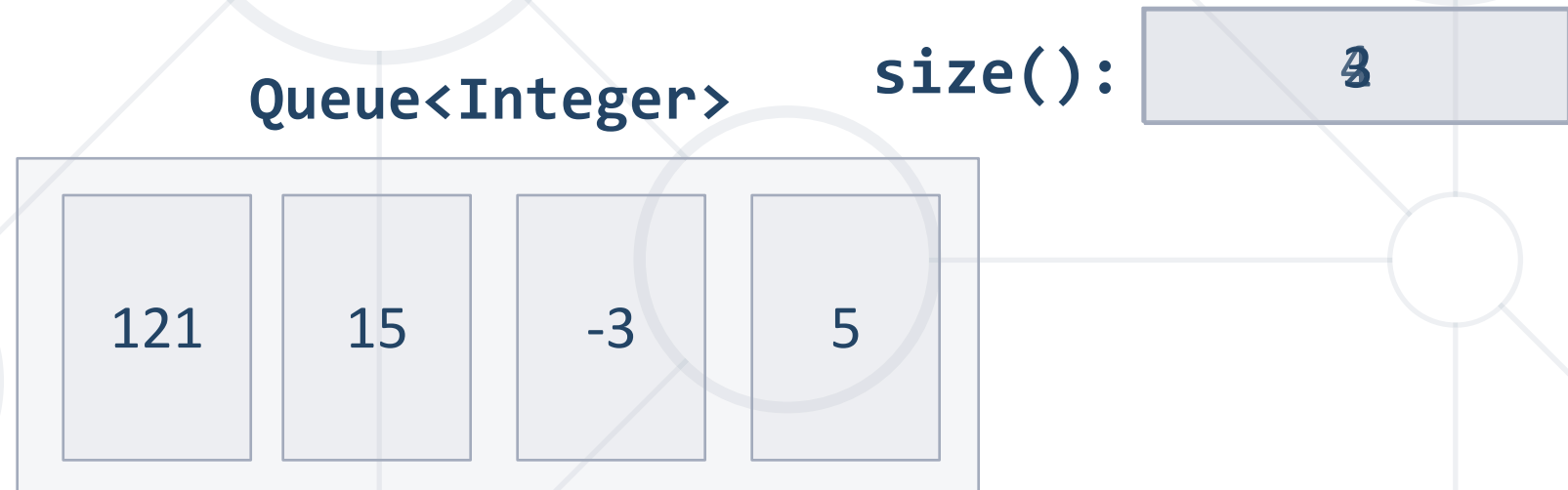
# Add() / offer()

- Adds an element to the queue



# Remove() / Poll()

- Returns and removes first element



# Problem: Hot Potato

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

| Input              |
|--------------------|
| Sam John Sara<br>2 |



| Output                                      |
|---|
| Removed John<br>Removed Sam<br>Last is Sara |

Check your solution here: <https://judge.softuni.bg/Contests/1437/Stacks-and-Queues-Lab>

# Solution: Hot Potato (1)

```
Scanner scanner = new Scanner(System.in);
String[] children = scanner.nextLine().split("\\s+");
int n = Integer.valueOf(scanner.nextLine());

ArrayDeque<String> queue = new ArrayDeque<>();

for (String child : children)
    queue.offer(child);

// continue...
```

# Solution: Hot Potato (2)

```
while (queue.size() > 1) {  
    for (int i = 1; i < n; i++)  
        queue.offer(queue.poll());  
  
    System.out.println("Removed " + queue.poll());  
}  
  
System.out.println("Last is " + queue.poll());
```

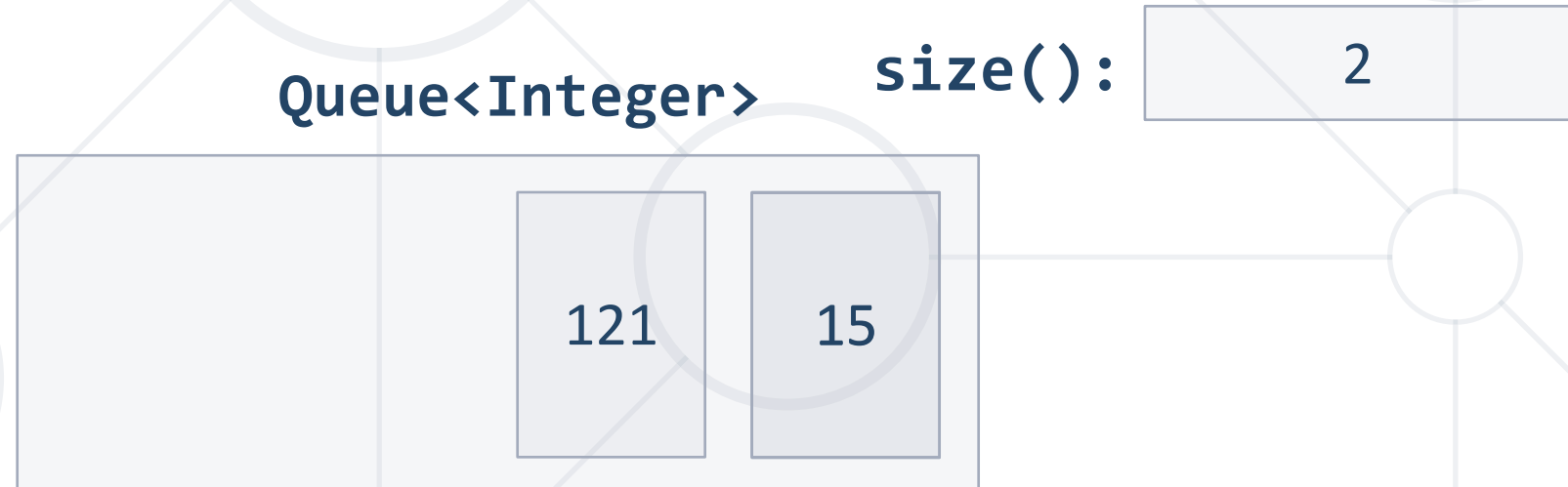
- Utility Methods

```
Integer element = queue.peek();  
Integer size = queue.size();  
Integer[] arr = queue.toArray();  
boolean exists = queue.contains(element);
```

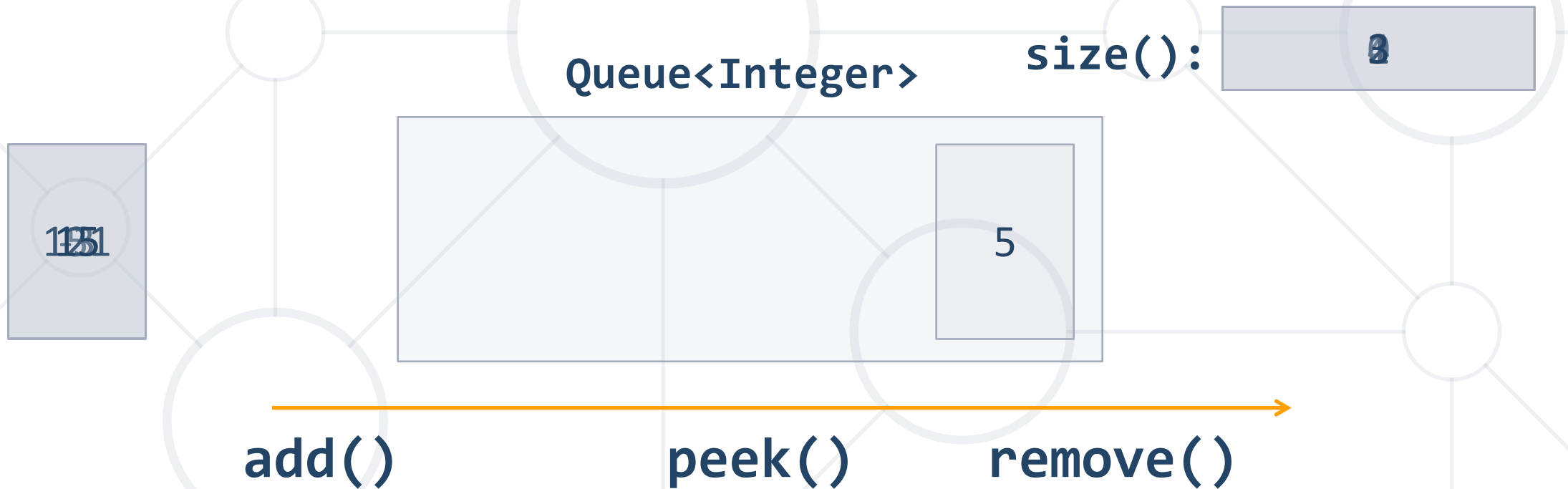
- **peek()** - checks the value of the first element
- **size()** - returns queue size
- **toArray()** - converts the queue to an array
- **contains()** - checks if element is in the queue

# Peek()

- Gets the first element without removing it

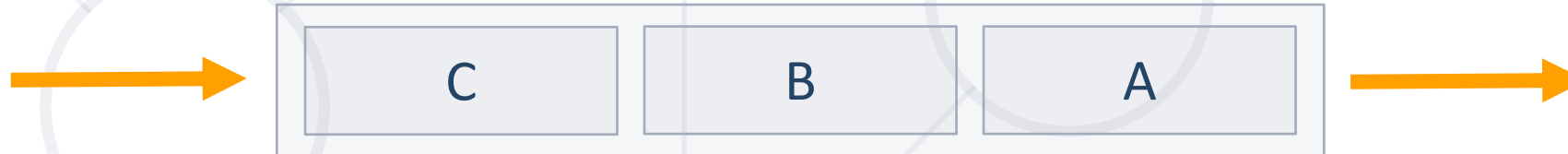


# Queue – Overview of All Operations





- Retains a specific order to the elements
- Higher priority elements are pushed to the beginning of the queue
- Lower priority elements are pushed to the end of the queue



# Problem: Math Potato

- Rework the previous problem by using a **PriorityQueue** so that a child is removed only on a prime cycle (cycles start from 1)
- If a cycle is not prime, just print the child's name

| Input                   |
|-------------------------|
| Maria Peter George<br>2 |



| Output   |
|--|
| Removed George<br>Prime Maria<br>Prime Maria<br>Removed Maria<br>Last is Peter |

# Solution: Math Potato

```
int cycle = 1;
while (queue.size() > 1) {
    for (int i = 1; i < n; i++)
        queue.offer(queue.poll());

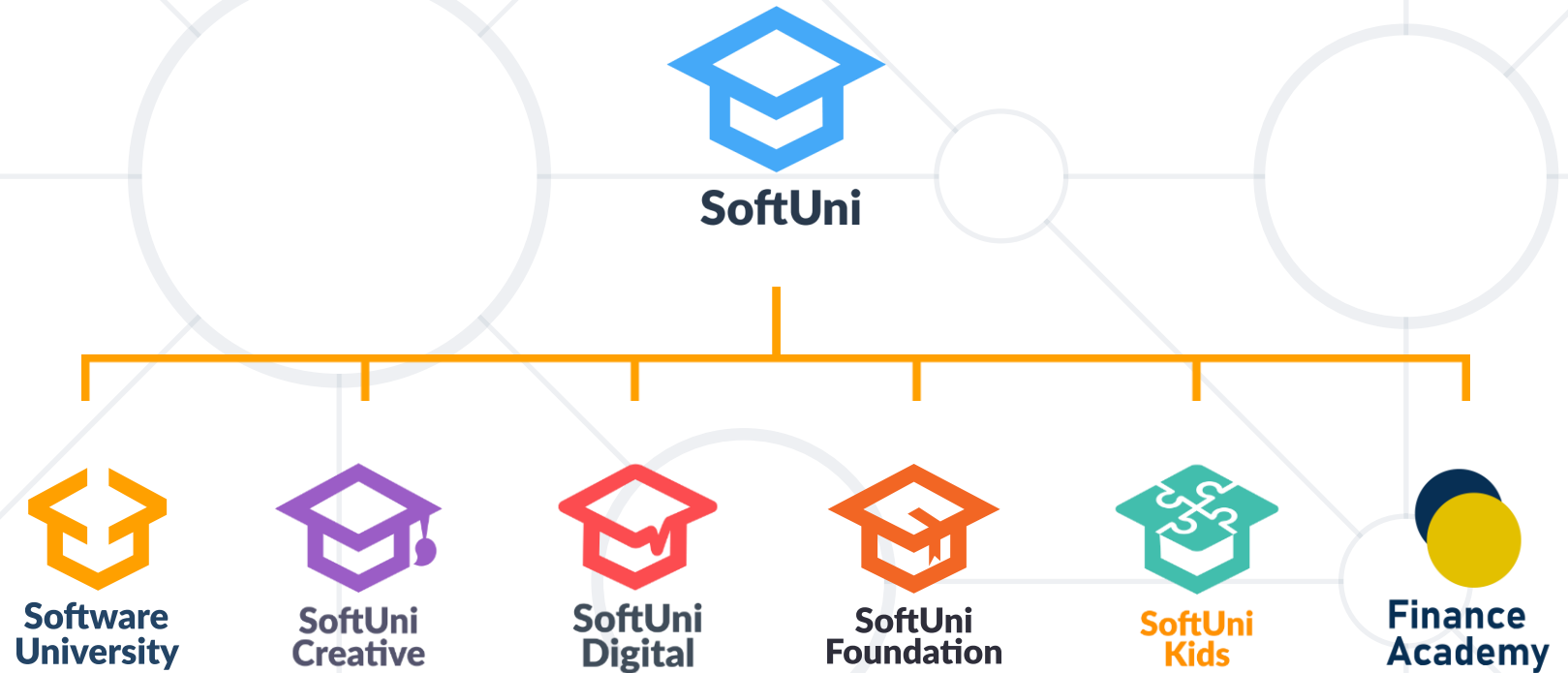
    if (isPrime(cycle))
        System.out.println("Prime " + queue.peek());
    else
        System.out.println("Removed " + queue.poll());

    cycle++;
}
System.out.println("Last is " + queue.poll());
```

- **Stack** - Last In First Out (LIFO)
  - push(), pop(), peek()
- **Queue** - First In First Out (FIFO)
  - add(), poll(), peek()
- **Priority Queue**



# Questions?



# SoftUni Diamond Partners

**SUPER  
HOSTING  
.BG**



**Coca-Cola HBC  
Bulgaria**



**POKERSTARS**  
POKER | CASINO | SPORTS  
a Flutter International brand

**INDEAVR**  
Serving the high achievers



**AMBITIONED**

 **DRAFT  
KINGS**



**SOFTWARE  
GROUP**

createX



**Postbank**  
Решения за твоето утре

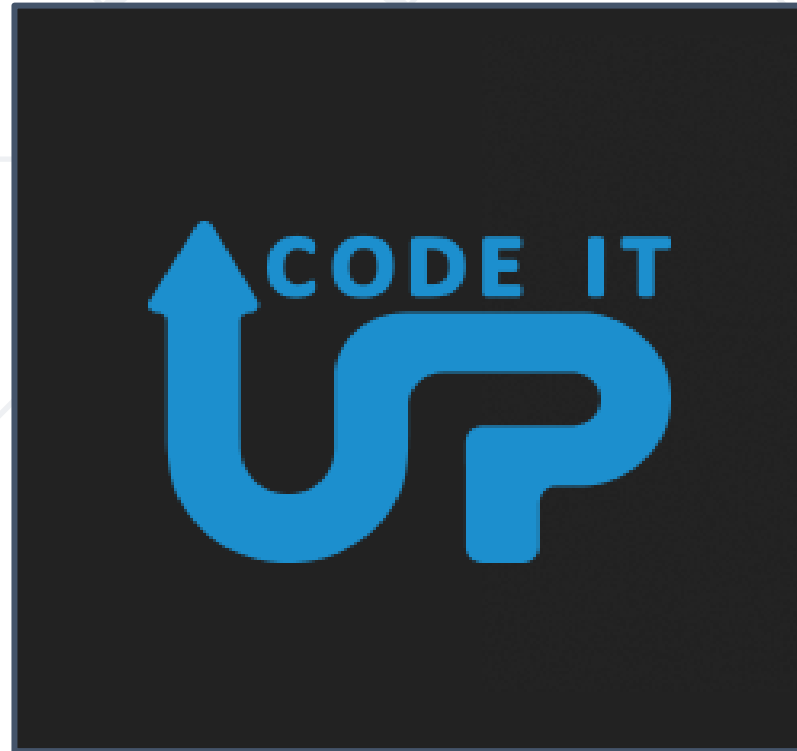


**BOSCH**

**DXC**  
TECHNOLOGY



**SmartIT**



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

- [softuni.bg](https://softuni.bg)

- Software University Foundation

- [softuni.foundation](https://softuni.foundation)

- Software University @ Facebook

- [facebook.com/SoftwareUniversity](https://facebook.com/SoftwareUniversity)

- Software University Forums

- [forum.softuni.bg](https://forum.softuni.bg)





- 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>

