**Bulk операция** – это термин, который часто используется в контексте обработки данных или выполнения операций в больших объемах или массовых операциях. Он обычно означает выполнение определенной операции одновременно или сразу с большим количеством элементов или записей данных, что обычно приводит к более эффективному использованию ресурсов и повышению производительности. Например, bulk операция может включать в себя массовую вставку, обновление или удаление данных в базе данных, массовую обработку файлов или любую другую операцию, выполняемую на большом объеме информации.

**Lambda-выражения** – это анонимные функции (может и не 100% верное определение для Java, но зато привносит некоторую ясность). Проще говоря, это метод без объявления, т.е. без модификаторов доступа, возвращающемого значения и имени.

**Stream API** - эффективный инструмент для обработки коллекций в функциональном стиле.

**Операции Stream API:**

Все методы **Stream API** можно разделить на две группы: промежуточные и терминальные. Промежуточные операции возвращают трансформированный поток. А терминальные операции возвращают конкретный результат.

1. **Промежуточные**

* **map() – преобразование в коллекцию.** Результатом может быть, например, список

Stream.of(1, 2, 3, 4, 5, 6)  
 .map(n -> n \* 10) // умножает каждый элемент на 10  
 Stream.of("apple", "orange")  
 .map(String::toUpperCase) // строки в верхний регистр

* **filter() – отсеивание части объектов**

Stream.of(1, 2, 3, 4, 5, 6)  
 .filter(n -> n > 3) // оставляет только те числа, которые больше 3

* **peek() – доступ к элементам без изменений**

Stream.of(1, 2, 3, 4, 5, 6)  
 .peek(System.out::println)

* **distinct() – оставляет только уникальные элементы**

Stream.of(2, 2, 3, 2, 3, 3, 2, 2)  
 .distinct() [2, 3]

* **sorted() – сортировка**

Stream.of(2, 1, 3, 5, 4, 6)  
 .sorted() // получим [1, 2, 3, 4, 5, 6]

* **limit() – ограничение по количеству элементов**

Stream.of(1, 2, 3, 4, 5, 6)  
 .limit(3) // [1, 2, 3]

* **skip() – пропуск первых элементов**

Stream.of(1, 2, 3, 4, 5, 6)  
 .skip(4) // [5, 6]

1. **Терминальные**

* **collect() – преобразование в коллекцию.** Результатом может быть, например, список.

List<String> fruits = Stream.of("apple", "banana", "lemon", "orange")  
 .collect(Collectors.toList());

* **forEach() – простой обход элементов**

Stream.of(5, 3, 2, 10, 8).forEach(System.out::println);

* **count() – аналог size()**

long count = Stream.of("apple", "banana", "lemon", "orange")  
 .count(); // 4

* **min() и max() – минимальное и максимальное значение**

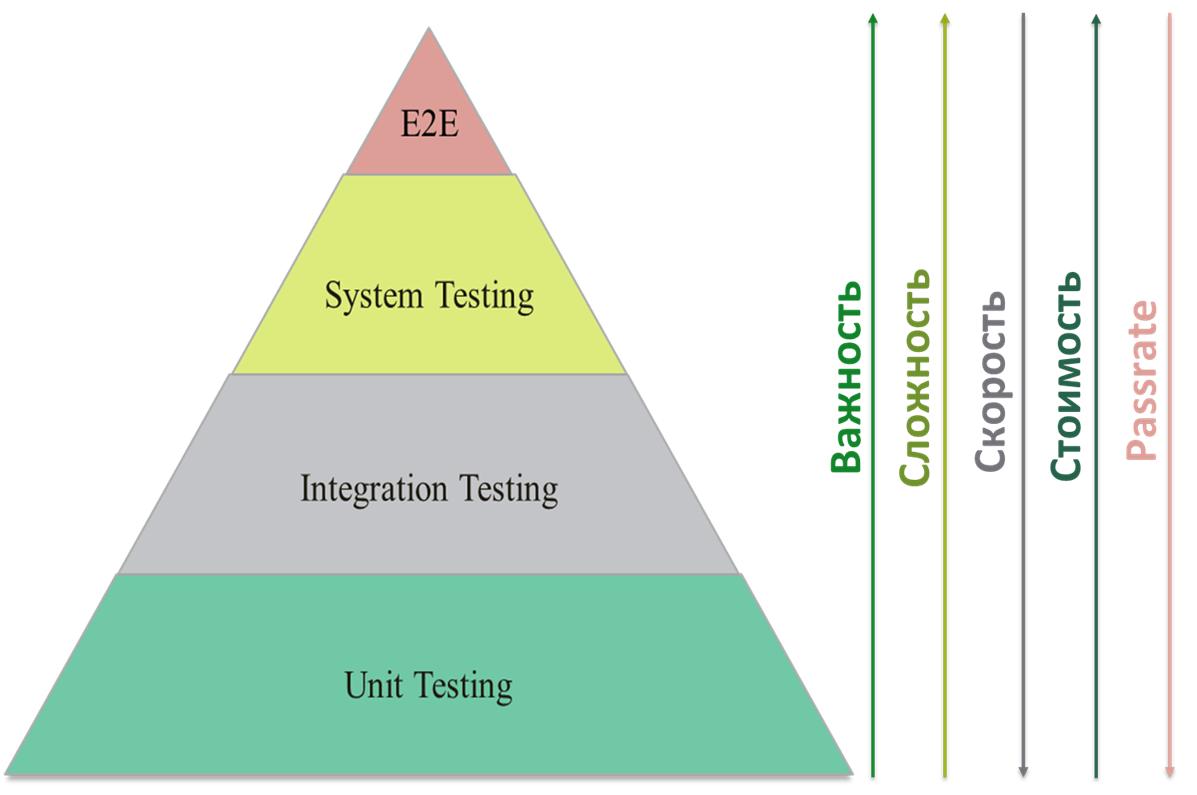
Supplier<Stream<Integer>> streamSupplier = () -> Stream.of(5, 3, 2, 10, 8);  
Optional<Integer> min = streamSupplier.get().min(Comparator.naturalOrder()); // [2]  
Optional<Integer> max = streamSupplier.get().max(Comparator.naturalOrder()); // [10]

* **allMatch(), noneMatch(), anyMatch() – соответствие всех элементов, ни одного или хотя бы одного.**

boolean isAllPositive = Stream.of(1, 2, 3, 4, 5)  
 .allMatch(e -> e > 0); // true  
  
boolean isAllPositive = Stream.of(1, 2, 3, 4, 5)  
 .noneMatch(e -> e < 0); // true  
  
boolean hasNegative = Stream.of(1, 2, 3, 4, 5)  
 .anyMatch(e -> e < 0); // false

**Виды тестирования:**

* Блочное (Unit testing) — тестирование одного модуля в изоляции.
* Интеграционное (Integration Testing) — тестирование группы взаимодействующих модулей.
* Системное (System Testing) — тестирование системы в целом.
* Приемочное (E2E) - **валидация требований** (проверка работы ПО в целом, не только по прописанным требованиям, что проверили на системном уровне).



**TDD**, test-driven development или процесс разработки через тестирование — это методология разработки программного обеспечения, которая основывается на повторении коротких циклов разработки:

1. Прежде чем писать код, пишут тест. Прежде чем переходить к следующему шагу, тест запускают и убеждаются в том, что он выдаёт ошибку. Благодаря этому можно быть уверенным в том, что тест не выдаёт ложноположительные результаты, это — своего рода тестирование самих тестов.
2. Пишут минимальную реализацию и добиваются того, чтобы она успешно прошла тестирование.
3. Выполняют, если это нужно, рефакторинг кода. Рефакторинг, при наличии теста, который способен указать разработчику на правильность или неправильность работы системы, вселяет в разработчика уверенность в его действиях.

**Unit тестирование VS Интеграционное тестирование**



**JUnit** и **Mockito** - это два популярных инструмента для тестирования в языке Java. JUnit используется для написания и запуска тестов, а Mockito - для создания и управления макетами (mock objects) в тестах.

import static org.junit.Assert.assertEquals;  
 import static org.mockito.Mockito.\*;  
  
 import org.junit.Test;  
  
public class CalculatorTest {  
  
 @Test  
 public void testAdd() {  
 // Создаем макет зависимости  
 MathService mathServiceMock = mock(MathService.class);  
  
 // Устанавливаем ожидаемое поведение макета  
 when(mathServiceMock.add(2, 3)).thenReturn(5);  
  
 // Создаем экземпляр класса, который будем тестировать  
 Calculator calculator = new Calculator(mathServiceMock);  
  
 // Вызываем метод, который тестируем  
 int result = calculator.add(2, 3);  
  
 // Проверяем, что результат соответствует ожидаемому  
 assertEquals(5, result);  
  
 // Проверяем, что метод макета был вызван с определенными аргументами  
 verify(mathServiceMock).add(2, 3);  
 }  
}