Вопросы:

21. Обработка исключений. Объявление исключений. Привести пример (представить код).

**Ответ:**

Объявление исключений происходит с использованием ключевого слова throws. Оно указывает, что метод может выбросить исключение, которое нужно обработать вызывающему коду. Это позволяет компилятору проверять, обработаны ли исключения (checked exceptions).

import java.io.IOException;

public class ExceptionExample {

// Метод объявляет, что может выбросить IOException

public void readFile(String filePath) throws IOException {

if (filePath == null) {

throw new IOException("Путь к файлу не может быть пустым.");

}

// Логика чтения файла

System.out.println("Файл успешно прочитан.");

}

public static void main(String[] args) {

ExceptionExample example = new ExceptionExample();

try {

example.readFile(null);

} catch (IOException e) {

System.out.println("Исключение перехвачено: " + e.getMessage());

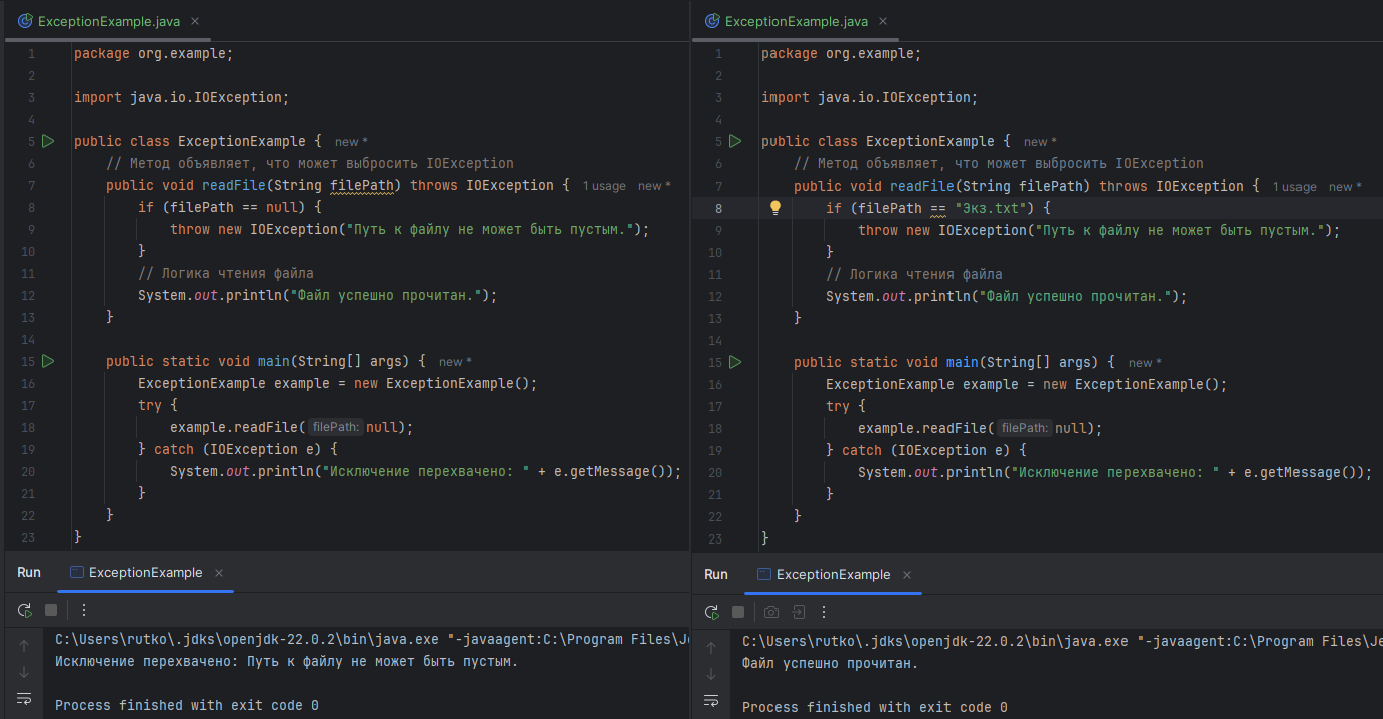
}

}

}

**Объяснение:**

1. Метод readFile объявляет, что он может выбросить исключение IOException (проверяемое исключение).
2. Если переданный аргумент filePath равен null, выбрасывается IOException.
3. В методе main вызывается метод readFile, но перед этим используется блок try-catch для обработки возможного исключения.
4. При передаче null в качестве аргумента исключение выбрасывается и обрабатывается блоком catch, который выводит сообщение: Исключение перехвачено: Путь к файлу не может быть пустым.

**Работа кода:**  


22. Обработка исключений. Выбрасывание исключений. Привести пример (представить код).

**Ответ:**

Исключения выбрасываются с использованием ключевого слова throw. Это действие инициирует исключение в момент выполнения программы. Часто используется для обработки ошибок, которые нельзя исправить локально. Обычно применяется, если в процессе выполнения программы обнаруживается ошибка, требующая завершения текущей операции.

public class ThrowExample {

public void divide(int a, int b) {

if (b == 0) {

throw new ArithmeticException("Деление на ноль невозможно.");

}

System.out.println("Результат: " + (a / b));

}

public static void main(String[] args) {

ThrowExample example = new ThrowExample();

example.divide(10, 2);

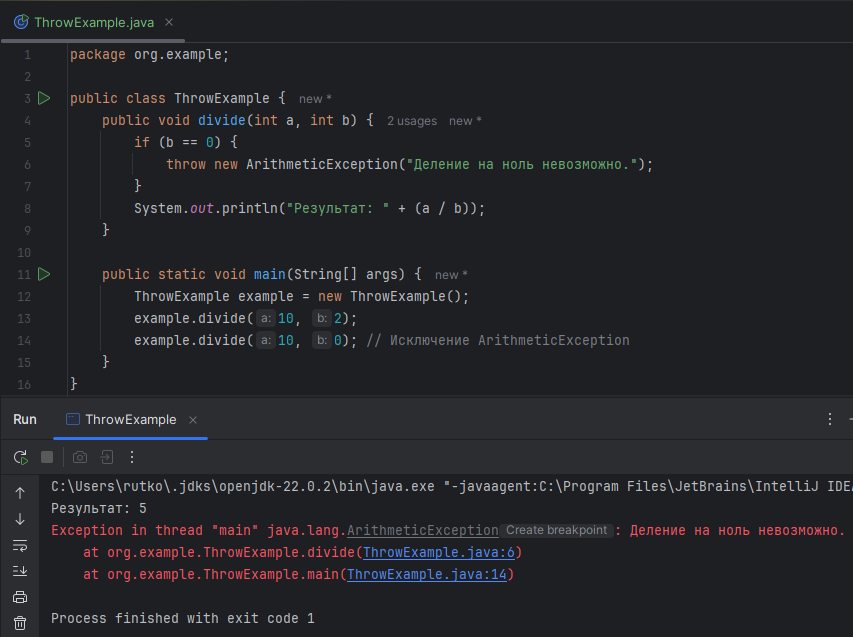
example.divide(10, 0); // Исключение ArithmeticException

}

}

**Объяснение:**

1. Метод divide проверяет, равен ли делитель нулю.
2. Если делитель равен нулю, выбрасывается исключение ArithmeticException с сообщением " Деление на ноль невозможно.".
3. Вызов example.divide(10, 2) работает корректно и выводит результат деления: "Результат: 5".
4. Вызов example.divide(10, 0) выбрасывает исключение, так как деление на ноль недопустимо.



23. Обработка исключений. Перехват исключений. Привести пример (представить код).

**Ответ:**

Перехват исключений осуществляется с использованием конструкции try-catch. Код, который может выбросить исключение, помещается в блок try, а обработка исключений — в блоке catch.

public class CatchExample {

public static void main(String[] args) {

try {

int[] numbers = {1, 2, 3};

System.out.println(numbers[5]); // Попытка доступа к несуществующему элементу

} catch (ArrayIndexOutOfBoundsException e) {

System.out.println("Обработано исключение: " + e.getMessage());

}

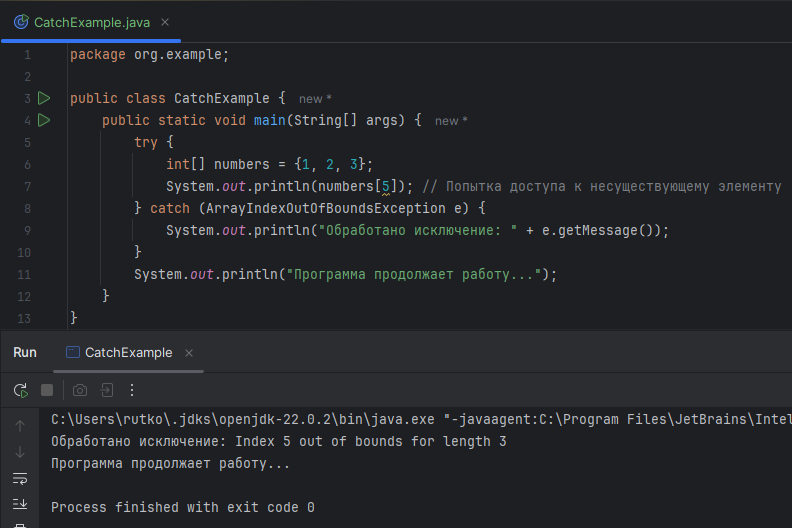
System.out.println("Программа продолжает работу...");

}

}

**Объяснение:**

1. В блоке try осуществляется попытка доступа к элементу массива по несуществующему индексу 5.
2. Исключение ArrayIndexOutOfBoundsException выбрасывается и перехватывается блоком catch.
3. Сообщение об исключении выводится, но программа продолжает выполнение.



24. Обработка исключений. Получение информации об исключениях. Привести пример (представить код).

**Ответ:**

При обработке исключения можно получить информацию о его типе, сообщении и стеке вызовов. Для этого используются методы getMessage(), printStackTrace() и getCause().

Методы:

* getMessage() возвращает описание ошибки.
* printStackTrace() выводит стек вызовов.
* getClass().getName() возвращает имя класса исключения.

public class ExceptionInfoExample {

public static void main(String[] args) {

try {

String str = null;

System.out.println(str.length()); // Попытка вызова метода у null

} catch (NullPointerException e) {

System.out.println("Тип исключения: " + e.getClass().getName());

System.out.println("Сообщение: " + e.getMessage());

System.out.print("Стек вызовов: ");

e.printStackTrace();

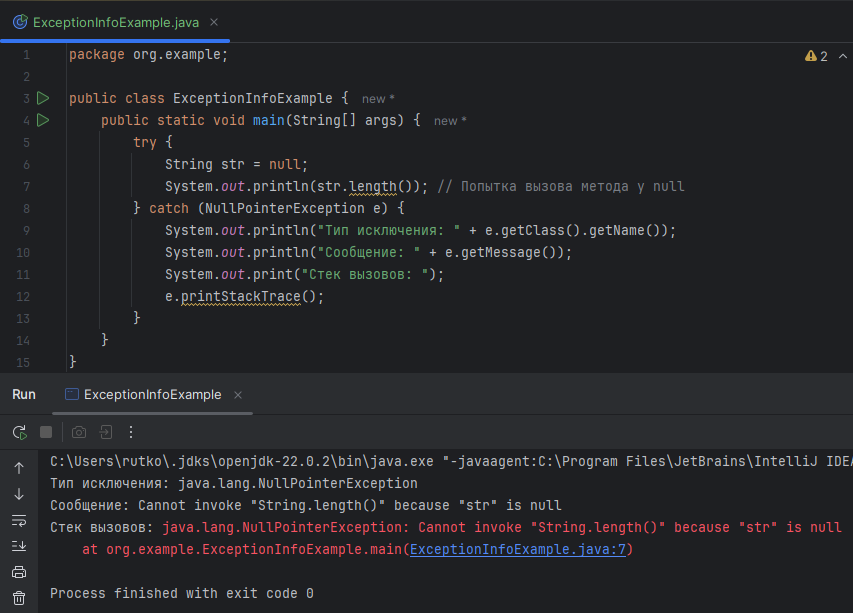
}

}

}

**Объяснение:**

1. Вызов метода length() у null выбрасывает NullPointerException.
2. В блоке catch выводится тип исключения, сообщение и стек вызовов.
3. Это полезно для диагностики проблем в коде.



25. Обработка исключений. Когда следует использовать исключения. Привести пример (представить код).

**Ответ:**

Исключения используются для обработки ошибок, которые невозможно предсказать или избежать при помощи условий. Они помогают отделить код обработки ошибок от основной логики программы.

import java.util.Scanner;

public class ExceptionUsageExample {

public static int divide(int a, int b) throws ArithmeticException {

if (b == 0) {

throw new ArithmeticException("Деление на ноль недопустимо.");

}

return a / b;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

try {

System.out.print("Введите числитель: ");

int numerator = scanner.nextInt();

System.out.print("Введите знаменатель: ");

int denominator = scanner.nextInt();

System.out.println("Результат: " + divide(numerator, denominator));

} catch (ArithmeticException e) {

System.out.println("Ошибка: " + e.getMessage());

} finally {

scanner.close();

System.out.println("Сканер закрыт.");

}

}

}

**Объяснение:**

1. Метод divide проверяет делитель. Если делитель равен нулю, выбрасывается ArithmeticException.
2. Пользователь вводит числа через консоль. Если делитель равен нулю, исключение обрабатывается, а программа продолжает работать корректно.
3. В блоке finally выполняются действия, которые должны быть завершены независимо от успешного или аварийного выполнения программы (например, закрытие ресурса scanner).

