ИСКЛЮЧЕНИЯ

1. ***Что такое исключение.***

**это событие, которое происходит во время выполнения (компеляции) программы и нарушает нормальный ход её выполнения. Когда возникает исключение оно создает объект.**

1. Иерархия исключений.

**Родительский класс – Throwable**

**Throwable делится на два основных подкласса: Error и Exception.**

**Exception делится на:**

** Проверяемые исключения (checked exceptions): Должны быть либо обработаны с помощью блока try-catch, либо объявлены в сигнатуре метода с использованием ключевого слова throws.**

** Непроверяемые исключения (unchecked exceptions): Подклассы RuntimeException и Error. Эти исключения могут возникать в любое время, и их обработка не обязательна.**

1. ***От каких классов Throwable и его подклассов нельзя наследоваться?***

**Throwable - нарушает концептуальную целостность иерархии исключений, не будет распознано как часть иерархии (Exception) или ошибок (Error), что может привести к неправильной обработке в catch-блоках.**

**Error (указывают на проблемы уровня JVM или системные ошибки, и их обработка вне контроля приложения.)**

1. ***Какое назначение класса Throwable? Методы класса Throwable***

**предоставляя методы для получения информации об исключениях и управления потоком выполнения программы.**

**getMessage() Возвращает сообщение**

**toString() - Возвращает краткое описание этого исключения**

**printStackTrace() - Печатает трассировку стека этого исключения на стандартный поток ошибок**

**getCause() - Возвращает причину этого исключения или null, если причина отсутствует или неизвестна.**

**StackTraceElement[] getStackTrace() - Возвращает массив элементов трассировки стека, которые показывают точки, в которых было выброшено исключение**

1. **Чем отличаются исключения от обычных классов?**

**Назначению:**

**Обычные классы**: Созданы для представления различных сущностей и логики приложения

**Исключения**: Созданы для представления ошибок и исключительных ситуаций

**Иерархия**

**Обычные классы**: Могут наследоваться от любых классов, кроме классов, связанных с исключениями

**Исключения**: Наследуются от класса Throwable или его подклассов (Exception и Error).

**Специальные методы и поля**

**Обычные классы**: Могут содержать любые поля и методы, необходимые для бизнес-логики.

**Исключения**: Имеют доступ к специальным методам и полям, которые предоставляются классом Throwable

**Создание и выбрасывание**

**Обычные классы**: Создаются и используются по мере необходимости для реализации бизнес-логики.

**Исключения**: Создаются и выбрасываются (throw) в случае возникновения ошибок или исключительных ситуаций

**Обработка**

**Обычные классы**: Не имеют специальных механизмов обработки.

**Исключения**: Обрабатываются с помощью блоков try, catch, finally

1. ***Как создать/бросить/поймать исключение.***

**создаются путем наследования от класса Exception или RuntimeException**

**Чтобы бросить исключение, используйте ключевое слово throw**

**Исключения перехватываются с помощью блоков try-catch**

1. ***Может ли main выбросить исключение?***

**может**

1. **Зачем создавать свой класс и наследовать его от Exception?**

**Упрощение обработки ошибок** - **Собственные исключения позволяют более четко разграничить и идентифицировать типы ошибок, которые могут возникать в приложении**

**Улучшение читаемости и поддержки кода - создаете исключение, которое описывает конкретную ошибку(**UserNotFoundException)

**Предоставить больше информации об ошибке** - **дополнительные поля и методы в своё исключение для хранения дополнительной информации, связанной с ошибкой**

1. ***В чем разница между checked и unchecked исключениями.***

**Checked – при копиляции, компелятор заставляет проверять это исключение (Саб классу Extention) мы должны обрабатывать tru/catch или писать метод может выбросить checked exeption - throw**

**Unchecked (Runtime) – при выполнении. ( Саб класс RuntimeExtention)**

1. **Можно ли обработать непроверяемые исключения?**

Можно.

**обработка позволяет предотвратить неконтролируемое завершение программы и помогает в исправлении ошибок**.

**позволяет вам записывать информацию о возникших ошибках, что полезно для отладки и логирования.**

1. ***Можно ли в условии throws указать непроверяемое исключение?***

**можно указать непроверяемое исключение (unchecked exception), хотя это и не обязательно.**

1. ***Что такое стектрейс. Какую информацию из него можно получить?***

**Показывает «дерево» откуда и куда приходил Exception. И понять кто был источником выброса ошибки**

1. ***Какую информацию можно получить из StackTraceElement?***

**содержит информацию о текущем методе, классе и файле, а также строке, где произошла ошибка**

**getClassName - возвращает имя класса**

**getMethodName- возвращает метод класса**

**getFileName - Возвращает имя исходного файла или null если файл недоступен**

**getLineNumber - Возвращает номер строк или -1 если номер строки не доступен**

1. ***Можно ли после try написать несколько catch? Правило.***

**Можно**

**Блоки catch должны быть упорядочены от более конкретных исключений к более общим.**

**Каждый блок catch может обрабатывать один тип исключения или группу исключений, чтобы обрабатывать различные исключения по-разному.**

1. ***Можно ли в одном блоке catch обработать несколько исключений?***

**Да, через оператор « | » - или.**

1. ***блок finaly.***

**будет выполнен независимо от того, произошло исключение или нет**

1. ***4 случая когда не выполнится finally***

**Если JVM (Java Virtual Machine) завершает работу - System.exit(0)**

**Если в блоке finally происходит исключение, которое не обрабатывается**

**Прерывание потока: Если поток был прерван с помощью метода Thread.interrupt()**

**Системный сбой**

1. ***Будет ли выполнен finally при Error?***

будет

1. ***Можно ли так написать try { throw new Object(); }*** ?

**Можно, но это бессмысленно.**

**Объект класса Object не предоставляет никакой информации о типе ошибки или ее возникновения.**

**Object не предоставляет методов или полей для описания ошибок,**

**В блоке catch тип исключения должен соответствовать или быть суперклассом типа исключения**

1. ***Конструкция try-catch-with-resource.***

**Конструкция try-with-resources в языке Java предназначена для автоматического управления ресурсами, такими как файлы, сокеты и базы данных. избавляя от необходимости явно закрывать ресурсы в блоке finally**

1. ***Что такое ресурс в конструкции try-with-resources?***

**это объект, который реализует один из интерфейсов, поддерживающих автоматическое закрытие ресурсов**

1. ***Try с ресурсами: что если исключение вылетело сначала в try, а потом в close в конструкции try-with-recources? Какое исключение вылетит? Что будет с другим?***
2. ***2 исключения, 1 в try, другое при закрытии ресурсов. Какое увидим?***

**приоритетное исключение будет то, которое возникло в блоке try. Второе исключение, возникшее в методе close, будет добавлено как подавленное (suppressed exception) к первому исключению.**

1. ***Когда происходит закрытие ресурсов в try-with-recources?***

**закрытие ресурсов происходит автоматически в конце блока try, когда выполнение достигает конца блока, либо когда управление покидает блок try из-за выброшенного исключения или оператора управления потоком (например, return, break, или continue).**

1. ***2 return, 1 в try, другой в finally. Какой не выполнится?***
2. ***2 исключения, одно в try, другое в finally. Какое увидим?***

**В блоке finaly, т.к он имеет больший приоритет.**

1. ***Что такое подавленные исключения? Как достать подавленное исключение?***

 **Подавленные исключения сохраняются вместе с основным исключением и могут быть получены с помощью метода getSuppressed.**

** Подавленные исключения используются для предоставления дополнительной информации о проблемах, возникших при закрытии ресурсов, не теряя при этом основное исключение, возникшее в блоке try.**

1. ***Что такое ошибка, а что такое исключительная ситуация?***

***Ошибки* — это проблемы в коде (синтаксические, логические, ошибки времени выполнения), которые обычно должны быть исправлены программистом.**

***Исключения* — это события, которые нарушают нормальное выполнение программы, но могут быть обработаны, чтобы программа могла корректно продолжить работу или завершиться**.

1. ***Нужно ли ловить Error?***

**Не рекомендуется ловить Error в Java, так как это указывает на серьезные проблемы, которые должны быть исправлены на уровне системы или окружения.**

ЛОГИРОВАНИЕ

34. ***Что такое логирование и для чего используется.***

**это процесс записи информации о работе программы в ходе ее выполнения. Эта информация записывается в лог-файлы или другие хранилища**

**Диагностика и отладка**

**Мониторинг и наблюдение**

**Аудит и безопасность**

**Отчетность и анализ**

35. ***Какие есть уровни логирования и для чего они нужны?***

**Trace - Используется для отслеживания выполнения программы на самом глубоком уровне.**

**Debug - Предназначен для детальной отладки программы. Пример: Входные параметры метода, промежуточные результаты вычислений**

**Info – Предоставляет информацию о нормальной работе системы. Пример: Запуск сервера, успешное завершение задачи**

**Warn – указывает проблемы, которые не требуют немедленного вмешательства. Пример: Использование устаревшего API.**

**Error - Указывает на серьезные ошибки. Пример: Ошибка подключения к базе данных.**

 **SEVERE**: Указывает на очень серьезные ошибки, которые могут привести к завершению программы.

 **WARNING**: Указывает на потенциальные проблемы или использование устаревших API.

 **INFO**: Информирует о нормальном ходе выполнения программы.

 **CONFIG**: Информирует о конфигурационных сообщениях.

 **FINE**: Предоставляет основную отладочную информацию.

 **FINER**: Предоставляет более детализированную отладочную информацию.

 **FINEST**: Предоставляет самую детализированную отладочную информацию.

 **ALL**: Включает все уровни логирования.

 **OFF**: Отключает логирование.

36. ***Чем отличаются FINE FINER FINEST?***

Пример: Вход и выход из метода.

Пример: Значения параметров метода и промежуточные результаты вычислений.

Пример: Пошаговое выполнение цикла или подробное логирование алгоритмов.

38. ***Сообщения каких уровней мы увидим, задав уровень INFO?***

**Если задан уровень INFO, то будут логироваться сообщения этого уровня и уровней, которые считаются более серьезными (высокими) по отношению к INFO.**

39. ***Обязательно ли передавать в метод getLogger() имя класса? Почему так принято?***

**Передача имени класса в getLogger() не является обязательной, это хорошая практика, которая улучшает читаемость, структурированность и управляемость логов.**