Глава 7 : Исключения, утверждения и протоколирование.

1.Исключение можно посмотреть в тетрадке.

2.Утверждения

1) Утверждения — широко распространенное средство так называемого безопасного программирования. Допустим, вы твердо знаете, что конкретное свойство уже задано, и обращаетесь к нему в своей программе. Например, при вычислении следующего выражения вы уверены в том, что значение параметра х не является отрицательным:

double у = Math.sqrt(х);

Разумеется, можно было бы просто сгенерировать исключение следующим образом:

if (х < 0) throw new IllegalArgumentException("х < 0");

Но у такого подхода имеется следующий существенный недостаток: приведенный выше код остается в программе даже после ее тестирования. Если в исходном коде программы расставлено много таких проверок, ее выполнение может замедлиться. Механизм утверждений позволяет вводить в исходный код программы проверки для ее тестирования, а затем удалять их из окончательного варианта. Для этой цели имеется ключевое слово assert. Ниже приведены две его основные формы:

assert условие;

assert условие : выражение;

Оба оператора проверяют заданное условиеи генерируют исключение типа AssertionError, если оно не выполняется. Во втором операторе выражение передается конструктору объекта типа AssertionError и преобразуется в символьную строку сообщения.

2) По умолчанию утверждения запрещены. Разрешить их в ходе выполнения программы можно с помощью параметра -enableassertions или -еа, указываемого в командной строке следующим образом:

java -enableassertions МуАрр

Кроме того, утверждения можно запретить в определенных классах и пакетах. Для этого достаточно указать параметр - diableassertions или -da в командной строке следующим образом:

java –еа : ... -da:MyClass MyApp

3) В языке Java предусмотрены следующие три механизма обработки системных сбоев. • Генерирование исключений.

• Протоколирование.

• Применение утверждений. Когда же следует применять утверждения? Чтобы правильно ответить на этот вопрос, необходимо учесть следующее.

• Утверждения оказываются ложными, если произошла неустранимая ошибка.

• Утверждения разрешаются только на время отладки и тестирования программ. (Иногда это положение шутливо формулируют так: надевайте спасательный жилет, плавая у берега, а в открытом море он вам все равно не поможет.)

Итак, утверждения не следует применять в качестве индикаторов несущественных и легко исправимых ошибок. Они предназначены для выявления серьезных неполадок во время тестирования.

3.Протоколирование

1)Программирующие на Java часто прибегают к вызовам метода System .out.println() , чтобы отладить код и уяснить поведение программы. Разумеется, после устранения ошибки следует убрать промежуточный вывод и ввести его в другом месте программы для выявления очередной неполадки. С целью упростить отладку программ в подобном режиме предусмотрен прикладной программный интерфейс API для протоколирования. Ниже перечислены основные преимущества его применения.

• Все протокольные записи нетрудно запретить или разрешить.

• Запрещенные протокольные записи отнимают немного ресурсов и не влияют на эффективность работы приложения.

• Протокольные записи можно направить разным обработчикам, вывести на консоль, записать в файл и т.п.

• Регистраторы и обработчики способны фильтровать записи. Фильтры отбрасывают ненужные записи по критериям, предоставляемым теми, кто реализует фильтры.

• Протокольные записи допускают форматирование. Их можно, например, представить в виде простого текста или в формате XML.

• В приложениях можно использовать несколько протоколов, имеющих иерархические имена, подобные именам пакетов, например com.mycompany.myapp.

• По умолчанию параметры настройки протоколирования задаются в конфигурационном файле. Такой способ задания параметров может быть изменен в приложении.

2) Как и имена пакетов, имена регистраторов образуют иерархию. На самом деле они являются еще более иерархическими, чем имена пакетов. Если между пакетом и его предшественником нет никакой семантической связи, то регистратор и производные от него регистраторы обладают общими свойствами. Так, если в регистраторе "com.mycompany" задать определенный уровень протоколирования, то производный от него регистратор унаследует этот уровень. Существует семь уровней протоколирования:

• SEVERE

• WARNING

• INFO

• CONFIG

• FINE

• FINER

• FINEST

По умолчанию используются первые три уровня, остальные нужно задавать, вызывая метод setLevel () следующим образом:

logger.setLevel(Level.FINE);

3) Ниже приведен "рецепт" организации протоколирования, в котором сведены наиболее употребительные операции.

1. Для простых приложений выбирайте один регистратор. Желательно, чтобы имя регистратора совпадало с именем основного пакета приложения, например com.mycompany.myprog. Создать регистратор можно, сделав следующий вызов:

Logger logger = Logger.getLogger("com.mycompany.myprog");

2. Для удобства в те классы, где интенсивно используется протоколирование, можно добавить статические поля:

private static final Logger logger = Logger.getLogger("com.mycompany.myprog");

3. По умолчанию все сообщения, имеющие уровень INFO и выше, выводятся на консоль. Пользователи могут изменить конфигурацию, предусмотренную по умолчанию, но, как пояснялось ранее, это довольно сложный процесс. Следовательно, лучше задать более оправданные настройки приложения по умолчанию. Приведенный ниже код гарантирует, что все сообщения будут зарегистрированы в файле протокола, связанном с конкретным приложением. Введите этот код в тело метода main () своего приложения.

if (System.getProperty("java.util.logging.config.class") == null && System.getProperty("java.util.logging.config.file") == null){

try {

Logger.getLogger("").setLevel(Level.ALL);

final int LOG\_ROTATION\_COUNT = 10;

Handler handler = new FileHandler("%h/myapp.log", 0, LOG\_ROTATION\_COUNT); Logger.getLogger("").addHandler(handler);

}

catch (IOException e) {

logger.log(Level.SEVERE, "Can't create log file handler", e);

}

}

4. Теперь все готово для протоколирования. Помните, что все сообщения, имеющие уровень протоколирования INFO, WARNING и SEVERE, выводятся на консоль. Следовательно, эти уровни протоколирования нужно зарезервировать для сообщений, представляющих ценность для пользователей вашей программы. Уровень FINE лучше выделить для сообщений, предназначенных для программистов. В тех местах кода, где вы обычно вызывали метод System.out. print In (), регистрируйте сообщения следующим образом:

logger.fine("File open dialog canceled");

5. Рекомендуется также регистрировать неожиданные исключения, например, так, как показано ниже.

try {

…}

catch (SomeException е) {

logger.log(Level.FINE, "explanation", e) ;

}