Exceptions

1. Множественные catch

Исключительные ситуации (исключения) возникают во время выполнения программы, когда появившаяся проблема не может быть решена в текущем контексте. При возникновении исключения в приложении создается объект, описывающий это исключение, текущий ход выполнения приложения останавливается, и включается механизм обработки исключений: управление передается соответствующему блоку catch, в котором он обрабатывается. Если в блоке try может быть сгенерировано в разных участках кода несколько типов исключений, то необходимо наличие нескольких блоков catch, если только блок catch не обрабатывает все типы исключений. Подклассы исключений в блоках catch должны следовать перед любым из их суперклассов, иначе суперкласс будет перехватывать эти исключения:

```
try {
    FileInputStream fileInputStream = new FileInputStream(fileName);
} catch (FileNotFoundException e) {
    System.err.println("fnfe");
} catch (IOException e) {
    System.err.println("ioe");
}
```

Если файл fileName не будет найден, то будет сгенерирован FileNotFoundException и выведено на консоль "fnfe". Если же файл будет найден, но, например, не доступен для записи, то будет сгенерирован IOException и не консоль будет выведено "ioe".

Бывают ситуации, когда в процессе выполнения программы могут возникнуть исключения разного типа, но их обработка ничем не отличается друг от друга:

```
try {
    //some code
} catch (ClassNotFoundException e) {
    e.printStackTrace();
} catch (NoSuchMethodException e) {
    e.printStackTrace();
} catch (FileNotFoundException e) {
    e.printStackTrace();
}
```

В Java 7 появилась возможность объединить такие исключения в одну конструкцию, чтобы не дублировать один и тот же код:

```
try {
    //some code
} catch (ClassNotFoundException | NoSuchMethodException | FileNotFoundException e) {
    e.printStackTrace();
}
```

2. Назначение оператора throws

Метод может генерировать исключения, которые сам не обрабатывает, а передает для обработки другим методам, вызывающим данный метод. В этом случае метод должен объявить о таком поведении с помощью ключевого слова throws с перечислением всех возможных типов исключений, чтобы вызывающий метод мог защитить себя от этих исключений. Это необходимо для всех исключений, кроме исключений типа Error, RuntimeException или любых их подклассов. В вызывающем методе должна быть предусмотрена или обработка этих исключений, или последующая передача выше, вызывающему методу. При этом этот метод сам может содержать блоки try-catch, а может

и не содержать их. Этот метод может выбрасывать исключение как "вручную" при помощи оператора throw, так и автоматически. Например:

```
private void printResult() {
    String[] userStrings = {"demo", ""};
    for (String userString : userStrings) {
        try {
            System.out.println(parseString(userString));
        } catch (IllegalArgumentException e) {
            System.err.println("iae");
        } catch (NumberFormatException e) {
            System.err.println("pe");
        }
    }
}

private int parseString(String userString) throws NumberFormatException,
        IllegalArgumentException {
    if (userString.isEmpty()) {
        throw new IllegalArgumentException();
    }
    int integerFromString = Integer.parseInt(userString);
    return integerFromString;
}

        /* Output:
        pe
        iae
```

Пример, конечно, искусственный и надуманный, но вполне демонстративный.
В методе parseString могут быть сгенерированы два типа исключений: NumberFormatException и IllegalArgumentException. Обработка этих исключений в этом методе не производится (хотя, конечно, может), а объекты исключений выбрасываются на уровень выше, то есть в метод printResult. При этом объект типа IllegalArgumentException выбрасывается вручную, а объект типа NumberFormatException автоматически (на самом деле точно так же вручную, но методом parseInt(String s, int radix), но для нашего метода parseString этот выбрал можно считать автоматическим).
В методе printResult происходит вызов parseString и обработка исключений, которые могут быть сгенерированы и выброшены из parseString.

---

3. Блок try с ресурсами

В Java 7 реализована возможность автоматического закрытия ресурсов в блоке try с ресурсами (try with resources). В операторе try открывается ресурс (файловый поток ввода), который затем читается. При завершении блока try данный ресурс автоматически закрывается, поэтому нет никакой необходимости явно вызывать метод close() у потока ввода, как это было в предыдущих версиях Java. Автоматическое управление ресурсами возможно только для тех ресурсов, которые реализуют интерфейс java.lang.AutoCloseable: InputStreamReader, FilterInputStream, FileReader и многие другие.

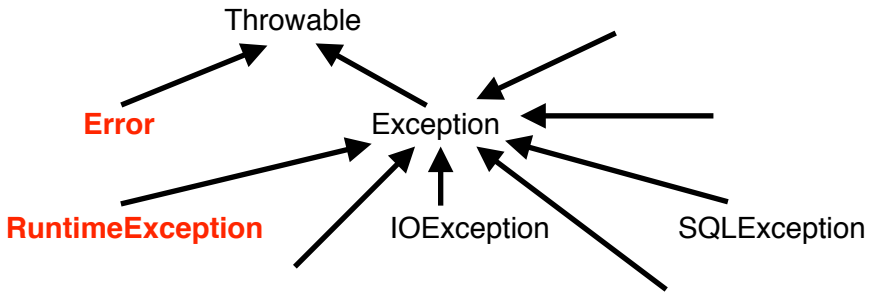
```
try (BufferedReader br = new BufferedReader(new InputStreamReader(System.in))) {
    System.out.println(br.readLine());
} catch (IOException e) {
    e.printStackTrace();
}
```

---

4. Checked и runtime исключения

Исключительные ситуации (исключения) возникают во время выполнения программы, когда появившаяся проблема не может быть решена в текущем контексте. При возникновении исключения в приложении создается объект, описывающий это исключение, текущий ход выполнения приложения останавливается, и включается механизм обработки исключений. При этом ссылка на объект-исключение передается обработчику исключений, который пытается решить возникшую проблему и продолжить выполнение программы.

Каждой исключительной ситуации поставлен в соответствие некоторый класс, экземпляр которого инициируется при исключительной ситуации. Если подходящего класса не существует, то он может быть создан разработчиком. Все исключения являются наследниками суперкласса Throwable и его подклассов **Error** и **Exception** из пакета java.lang.



Исключительные ситуации типа Error возникают только во время выполнения программы. Такие исключения связаны с серьезными ошибками уровня JVM, к примеру, с переполнением стека, не подлежат исправлению и не могут обрабатываться приложением.

Классы, наследуемые от Exception являются проверяемыми исключениями (checked), за исключением RuntimeException. Возможность возникновения проверяемого исключения отслеживается еще на этапе компиляции. Проверяемые исключения должны быть либо обработаны в методе, который их генерирует, либо выброшены на уровень выше и обработаны в вызывающем методе, за этим следит компилятор.

Класс RuntimeException и порожденные от него классы относятся к непроверяемым исключениям, компилятор не проверяет обработку этих исключений, они могут не обрабатываться. Такие исключения генерируются во время выполнения программы (в runtime) и связаны с ошибками программиста, например, недопустимое приведение типов, выход за пределы массива, деление целого числа на ноль и тд.

---

5. Повторный выброс исключения

При разработке кода возникают ситуации, когда в приложении необходимо инициировать генерацию исключения, например, для того, чтобы указать на некорректность параметров. Для генерации исключительной ситуации и ручного создания экземпляра исключения используется оператор throw. Экземпляром исключения может быть объект Throwable или его подклассов. Имеется два способа получения Throwable-объекта: использование параметра в предложении catch или создание объекта с помощью операции new:

```
1.
try {
   \\код, в котором может возникнуть исключение
} catch (ArrayIndexOutOfBoundsException e) {
   throw e;
}

2.
if (//условие) {
```

```
    throw new IllegalArgumentException();
}
```

При достижения оператора throw генерируется исключение, выполнение кода прекращается и ищется ближайший подходящий блок catch:

```
    private void throwEx(int a) throws IOException {
        if (a%2 == 0) {
            throw new IllegalArgumentException();
        } else {
            throw new IOException();
        }
    }

    private void printSmth() {
        try {
            throwEx(3);
        } catch (IllegalArgumentException e) {
            System.err.println("iae");
        } catch (IOException e) {
            System.err.println("ioe");
        }
    }
```

/* Output:
printSmth.catch

Если метод генерирует исключение с помощью оператора throw и при этом блок catch в методе отсутствует, как в методе throwEx() на примере выше, то для передачи обработки исключения вызывающему методу printSmth() тип проверяемого исключения должен быть указан в операторе throws при объявлении метода: private void throwEx(int a) throws IOException.

Для непроверяемых исключений, являющихся подклассами класса RuntimeException, как IllegalArgumentException на примере выше, throws в объявлении может отсутствовать, так как играет только информационную роль.

Перехваченное исключение может быть выброшено снова. Такая ситуация и называется повторным выбросом исключения. Например, блок catch отлавливает исключения типа IOException и выбрасывает их вновь.

```
    try{
        // some operations
    } catch(IOException e) {
        throw e;
    }
```

В Java 7 была введена возможность передавать «вверх» по стеку вызова исключение более точного типа, если данные типы указаны при объявлении метода.

```
public static void ioMethod() throws IOException {
    try{
        throw new IOException ("this is IOException ");
    } catch(Exception e) {
        throw e;
    }
}
```

6. Особенности работы блока finally

Возможна ситуация, при которой нужно выполнить некоторые действия по завершению программы вне зависимости от того, произошло исключение или нет. В этом случае используется блок finally, который обязательно выполняется после инструкций try или catch. Каждому блоку try должен соответствовать хотя бы один блок catch или finally. Блок finally обычно используется для освобождения ресурсов, захваченных при выполнении методы, например, для закрытия потоков и файлов. Этот блок выполняется перед выходом из метода даже если перед ним были выполнены такие инструкции, как return, break и тд:

```
private int getState() {
    try {
        System.out.println("try");
        return 1;
    } finally {
        System.out.println("finally");
        return 2;
    }
}

private void printState() {
    System.out.println(getState());
}


    /* Output:
    try
    finally
    2
```

7. Определение понятия исключения и исключительной ситуации. Оператор try-catch

Исключительные ситуации (исключения) возникают во время выполнения программы, когда появившаяся проблема не может быть решена в текущем контексте. При возникновении исключения в приложении создается объект, описывающий это исключение, текущий ход выполнения приложения останавливается, и включается механизм обработки исключений. При этом ссылка на объект-исключение передается обработчику исключений, который пытается решить возникшую проблему и продолжить выполнение программы. Исключение — это источник дополнительной информации о ходе выполнения приложения. Такая информация позволяет выявить ошибки на ранней стадии.

Каждой исключительной ситуации поставлен в соответствие некоторый класс, экземпляр которого инициируется при исключительной ситуации. Если подходящего класса не существует, то он может быть создан разработчиком.

Управление обработкой исключений в Java осуществляется с помощью пяти ключевых слов: try, catch, throw, throws и finally. Для задания блока программного кода, который требуется защитить от исключений, используется ключевое слово try. В этом блоке и генерируется объект исключения, если возникает исключительная ситуация, после чего управление передается в соответствующий блок catch. Блок catch помещается сразу после блока try, в нем задается тип исключения, которое требуется обработать. После блока try может быть несколько блоков catch для разной обработки разных исключений, может быть один блок catch, объединяющий разные исключения для их одинаковой обработки, а может и вообще не быть блока catch, но в этом случае обязательно должен быть блок finally. Общая форма обработки выглядит так:

```
try {
   \\код, в котором может возникнуть исключительная ситуация
} catch (тип_исключения_1 e) {
   \\обработка исключения 1
} catch (тип_исключения_2 e) {
   \\обработка исключения 2
} finally {
   \\действия, которые должны быть выполнены независимо от того, возникла
исключительная ситуация или нет
}
```

---

8. Ручная генерация исключений. Оператор throw

При разработке кода возникают ситуации, когда в приложении необходимо инициировать генерацию исключения, например, для того, чтобы указать на некорректность параметров. Для генерации исключительной ситуации и ручного создания экземпляра исключения используется оператор throw. Экземпляром исключения может быть объект Throwable или его подклассов. Имеется два способа получения Throwable-объекта: использование параметра в предложении catch или создание объекта с помощью операции new:

1.
```
try {
   \\код, в котором может возникнуть исключение
} catch (ArrayIndexOutOfBoundsException e) {
   throw e;
}
```

2.
```
if (//условие) {
   throw new IllegalArgumentException();
}
```

При достижения оператора throw генерируется исключение, выполнение кода прекращается и ищется ближайший подходящий блок catch:

```
   private void throwEx(int a) throws IOException {
      if (a%2 == 0) {
         throw new IllegalArgumentException();
      } else {
         throw new IOException();
      }
   }

   private void printSmth() {
      try {
         throwEx(3);
      } catch (IllegalArgumentException e) {
         System.err.println("iae");
      } catch (IOException e) {
         System.err.println("ioe");
      }
   }
```

/* Output:
printSmth.catch

Если метод генерирует исключение с помощью оператора throw и при этом блок catch в методе отсутствует, как в методе throwEx() на примере выше, то для передачи обработки исключения вызывающему методу printSmth() тип проверяемого исключения должен быть указан в операторе throws при объявлении метода: private void throwEx(int a) throws IOException.

Для непроверяемых исключений, являющихся подклассами класса RuntimeException, как IllegalArgumentException на примере выше, throws в объявлении может отсутствовать, так как играет только информационную роль.