6 Исключения

6.1 Определение

- Исключение в Java — это объект, который описывает исключительное состояние, возникшее в каком-либо участке программного кода.
- Когда возникает исключительное состояние, создается объект класса Exception.
- Этот объект пересылается в метод, обрабатывающий данный тип исключительной ситуации.
- По «следам» стека программы можно найти данный метод – и причину ошибки.

6.2 Возникновение исключения

Совершаем преднамеренную ошибку – делим на ноль.

```
class SimpleMistake {
      public static void main(String args[]) {
            int d = 0;
            int a = 42 / d;
      }
 }
```

- Тип исключения указывает на причину его возникновения.
- Стек вызовов позволяет отследить путь, по которому был достигнут проблемный код.
- Стандартный обработчик выдает номер строки кода, в котором произошло исключение.

6.3 Обработка исключения

```
try{
      doSomethingDangerous(); //опасный метод
   }
catch (CatchedExceptionType e) {
      treatDanger(); //обработка исключения
   }
//CatchedExceptionType – класс, к которому
      принадлежит исключение e
```

Ловим свою ошибку и выводим информацию на консоль.

```
class SimpleMistake{
     public static void main(String args[]){
          try{
               int d = 0; //выполнится
               int a = 42 / d;

               int z = a + d; //не выполнится
             }
          catch (ArithmeticException e) {
               System.out.println(«Деление на ноль");
             }
     }
}
```

6.4 Виды исключений

- Проверяемое
  - FileNotFoundException, IOException, …
  - После такой ситуации зачастую требуется восстановление состояния программы.
  - Обязательны для описания при определении метода.
- Ошибка
  - класс Error и его наследники.
- Исключение времени исполнения
  - Оно же непроверяемое.
  - RuntimeException и все наследники.
  - Восстановление после таких ситуаций обычно не производится.

6.5 Примеры исключений

ArithmeticException	ошибка при вычислениях – например, деление на 0
ArrayIndexOutOfBoundsException	выход за пределы массива
FileNotFoundException	если не обнаружен запрошенный файл
IOException	любое исключение в системе ввода/вывода; включает предыдущее
OutOfMemoryError	реакция на нехватку памяти
VirtualMachineError	ошибка внутри вируальной машины Java
AWTError	ошибка при работе графического интерфейса

6.6 Требования к коду

- Если метод вызывает проверяемое исключение, то он должен :
  - > либо обработать его
  - > либо передать исключение выше по стеку вызова
- Неудовлетворяющий этому правилу код не компилируется.

6.7 Каскад обработчиков

Иногда одного обработчика недостаточно – создаем несколько, на разные типы исключений.
В данном примере можно поймать сразу 2 исключения разного рода:

- Деление на ноль (при запуска программы без параметров),
- Выход за пределы массива (обращение к несуществующему 42-му элементу).

```
class MultiCatch {
    public static void main(String args[]) {
        try {
            int a = args.length;
            System.out.println("a = " + a);
            int b = 42 / a;
            int c[] = { 1 };
            c[42] = 99;
        } catch (ArithmeticException e) {
            System.out.println("div by 0: " + e);
        } catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("array index out of bound: " + e);
        }
    }
}
```

6.8 Вложенные блоки

Также можно вкладывать один блок try-catch в другой. Здесь в методе main вызывается метод procedure и обрабатывается ситуация деления на ноль. Попытка выхода за пределы массива обрабатывается уже внутри метода procedure() – во вложенном блоке try-catch.

```
class MultiNest {
    static void procedure() {
        try {
            int c[] = { 1 };
            c[42] = 99;
        } catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("array index out: " + e);
        }
}

    public static void main(String args[]) {
        try {
            int a = args.length();
            System.out.println("a = " + a);
            int b = 42 / a;
            procedure();
        } catch (ArithmeticException e) {
            System.out.println("div by 0: " + e);
        }
     }
 }
```

Этот метод сам обрабатывает свое же исключение, поэтому наружу исключение не передается.

```
  public static void doCalculation(){
    try {
          riskyCode();
    } catch(ArrayIndexOutOfBoundsException e) {
          tryToHandle();
     }
  }
```

6.9 Явно инициированное исключение

Новое исключение создается посредством вызова конструктора. Конструктор принимает строку, описывающую причину исключения. Генерирование исключения происходит с помощью оператора throw.

```
  class ThrowDemo {
     void riskyMethod(int value) {
       try {
         //какие-то действия
         if (value == 1){
           //бросаем исключение
            throw new IllegalArgumentException("Can't be 1");
         }
       }catch (IllegalArgumentException e) {
          prepareToClose();
         //передача исключения выше
          throw e;
       }
    }
```

6.10 Описание исключений

После имени метода указывается тип (типы) возможных исключений, которые метод может сгенерировать: throws.

```
class ThrowsDemo {
      static void riskyMethod() throws IllegalAccessException {
            //do something
            if (condition){
                  throw new IllegalAccessException("fake");
            }
      }
      public static void main(String args[]){
            riskyMethod();
      }
}
```

6.11 Блок finally

Используя данный блок, добиваемся того, что некий набор действий выполнится независимо от того, сгенерировано исключение или нет.

```
try {
      //какие-то действия
      doSomething();
      //иногда бросает исключение
      doSomethingRisky();
}catch (NumberFormatException e) {
    handleState(); //обрабатываем исключение
}finally {
      //действия, которые нужно выполнить независимо
      // от того, были ли сгенерировано исключение или нет
      doFinalStuff();
}
```

```
class FinallyDemo {
    static void exceptionDemoMethodA() {
        try {
            System.out.println("inside exceptionDemoMethodA");
            throw new RuntimeException("demo");
        }  finally {
            System.out.println("exceptionDemoMethodA's finally");
        }
    }
    static void exceptionDemoMethodB() {
        try {
           System.out.println("inside exceptionDemoMethodB");    return;
        } finally {
           System.out.println("exceptionDemoMethodB's finally");
   }
}
    public static void main(String args[]) {
        try {
         exceptionDemoMethodB();
         }catch (Exception e) {}
        exceptionDemoMethodB();
   }
}
```

6.12 Пользовательские классы-исключения

Создаем свой класс исключений на основе класса Exception.

```
class MyException extends Exception {
    private int detail;
    MyException(int a) {     detail = a:   }
    public String toString() {    return "MyException[" + detail + "]";   }
}

class ExceptionDemo {
    static void compute(int a) throws MyException {
        System.out.println("called computer + a + ").");
                if (a > 10)
                    throw new MyException(a);
        System.out.println("normal exit.");
    }
```

Используется обычным способом.

```
    public static void main(String args[]) {
        try {
            compute(1);
            compute(20);
        } catch (MyException e) {
            System.out.println("caught" + e);
        }
   }
}
```

7 Работа со строками

7.1 Пример метода toString()

Каждый класс реализует метод toString (поскольку это метод класса Object).
Реализация определяет строковое представление объекта класса.
Можно переопределить данный метод и задать собственное, «говорящее» строковое представление – в данном случае выводятся на печать координаты точки.

```
class Point {
    int x, y;
    Point(int x, int y) {     this.x = x;     this.y = y;   }
    public String toString() {
        return "Point[" + x + ", " + y + "]";
    }
}
class ToStringDemo {
    public static void main(String args[]) {
        Point p = new Point(10, 20);
        System.out.println("p = " + p);
    }
 }
```

7.2 Поэлементный доступ

Существует ряд методов, позволяющих обращаться к элементам строки – символам (char).
В примере сначала в цикле выводятся символы строки s (обращение к ним идет при помощи метода charAt). Затем выводится подстрока строки s, для которой указаны индексы начального и конечного символов (end и start).

```
class GetCharsDemo {
    public static void main(String args[]) {
        String s = "This is a demo of the getChars method.";
        int start = 10;
        int end = 14;
        for (int i=0; i< s.length(); i++){
            System.out.println(s.charAt(i));
        }
        char buf[] = new char[end - start];
        s.getChars(start, end, buf, 0);
        System.out.println(buf);
    }
}
```

7.3 Сравнение строк

Строки можно сравнивать при помощи метода equals – он вернет true если строки одинаковы. Если регистр букв не должен учитываться при сравнении, следует использовать equalsIgnoreCase.

```
class EqualDemo {
    public static void main(String args[]) {
        String s1 = "Hello";
        String s2 = "Hello";
        String s3 = "Good-bye";
        String s4 = "HELLO";
        System.out.println(s1 + " equals " + s2 + " -> " + s1.equals(s2));
        System.out.println(s1 + " equals " + s3 + " -> " + s1.equals(s3));
        System.out.println(s1 + " equals " + s4 + " -> " + s1.equals(s4));
        System.out.println(s1 + " equalsIgnoreCase " + s4 + " -> " +
                           s1.equalsIgnoreCase(s4));
    }
}
```

7.4 Оператор равенства

Следует быть осторожным при использовании оператора равенства в случае строк (поскольку строки являются ссылочными типами). В приведенном примере результат первого сравнения – true, а второго – false.

```
class EqualsNotEqualTo {
    public static void main(String args[]) {
        String s1 = "Hello";
        String s2 = new String(s1);
        System.out.println(s1 + " equals " + s2 + " -> " + s1.equals(s2));
        System.out.println(s1 + " == " + s2 + ", -> " + (s1 == s2));
    }
}
```

7.5 Сравнение и сортировка строк

Здесь массив строк сортируется в алфавитном порядке. Строки сравниваются при помощи метода compareTo(). При необходимости меняются местами (сортировка «пузырьком»).

```
class SortString {
  static String arr[] = {"Now", "is", "the", "time", "for", "all",
                         "good", "men", "to", "come", "to", "the",
                         "aid", "of", "their", "country" };
  public static void main(String args[]) {
    for (int j = 0; j < arr.length; j++) {
      for (int i = j + 1; i < arr.length; i++) {
          if (arr[i].compareTo(arr[j]) < 0) {
              String t = arr[j];
              arr[j] = arr[i];
              arr[i] = t;
          }
      }
      System.out.println(arr[j]);
      }
   }
}
```

7.6 StringBuffer

Строка – неизменяемый объект, соответственно при каждой операции вставки, конкатенации и т.п. создаются новые объекты. Для интенсивной работы со строками более подходят классы StringBuffer и StringBuilder.

```
class StringBufferDemo {
    public static void main(String args[]) {
        StringBuffer sb = new StringBuffer("Hello");
        System.out.println("buffer = " + sb);
        System.out.println("length = " + sb.length());
        System.out.println("capacity = " + sb.capacity());
    }
 }
```

8 Многопоточные приложения

Поток - последовательность выполняемых операций внутри программы. Потоки позволяют исполнение нескольких действий одновременно. Поддержка многопоточного программирования встроена в язык Java.

Поток может находиться в одном из следующих состояний.

- New – создан, но не запущен.
- Running – активно использует ресурс CPU.
- Blocked – ожидает ресурсов или событий.
- Resumed – готов к работе после остановки типа block или suspend.
  - > Поток помещается в очередь, ожидающую доступа к ресурсам ЦП.
- Suspended – поток перестал использовать ЦП.
  - > Самостоятельно или был принудительно снят с исполнения.

8.1 Класс java.lang.Thread

Данный класс предоставляет абстракцию, позволяющую выполнять инструкции языка Java в отдельном потоке. Имеет следующие конструкторы.

Thread()	Создает новый класс.
Thread(String name)	Создает новый именованный класс-поток.
Thread(Runnable target)	Создает новый класс-поток на основе экземпляра Runnable, чей метод run() будет исполняться в отдельном потоке.
Thread(Runnable target, String name)	Создает новый именованный класс-поток на основе Runnable.

Среди методов класса можно выделить:

public static Thread currentThread()	Возвращает ссылку на активный в настоящее время поток.
public void interrupt()	Прерывает исполнение потока.
public final boolean isAlive()	Определяет, работает ли поток.
public void run()	Метод запускается в отдельном потоке.
public void start()	Запускает выполнение метода run() в отдельном потоке.
public void join(Thread other)	Приостанавливает исполнение текущего потока до тех пор пока поток other не завершится.
public void stop()	Завершает выполнение потока.
public void resume()	Восстанавливает выполнение потока.
public void sleep(long delay)	Приостанавливает выполнение потока на delay миллисекунд.
public void yield()	Метод пробует отказаться от выполнения на ЦП в пользу других потоков.

Класс Thread имеет несколько публичных полей, отражающих относительные приоритеты исполнения: **MAX_PRIORITY, MIN_PRIORITY и NORM_PRIORITY.**

Создать новый поток можно одним из следующих способов:

- Наследование от класса Thread

Наследник класса Thread переопределяет метод run(). Вызов метода start() у экземпляра подкласса запускает выполнение потока. Операторы, помещенные в методе run() выполняются виртуальной машиной в новом потоке.

- Реализация интерфейса **java.lang.Runnable**

Интерфейс Runnable должен быть реализован классом, который должен выполняться в отдельном потоке. Для этого класс должен реализовывать всего один метод - run().. Экземпляр класса должен быть передан конструктору класса Thread в качестве аргумента. Запуск потока осуществляется методом start() класса Thread.

8.2 Пример многопоточного приложения на основе класса Thread

Следующий пример выводит на консоль сообщения от двух потоков исполнения. Можно отметить, что потоки выполняются одновременно.

```
public class ConcurrentDemo{
    public static void main(String []args){
        new ConcurrentThread1().start();
        new ConcurrentThread2().start();
    }
}
class ConcurrentThread1 extends Thread{
    public void run(){
        for (int i=0;i<=10000;i++) System.out.println("#1: "+i);
    }
}
class ConcurrentThread2 extends Thread{
    public void run(){
        for (int i=0;i<=10000;i++) System.out.println("#2: "+i);
    }
}
```

8.3 Пример создания потока методом реализации интерфейса Runnable

```
class SimpleCombat implements Runnable{
    public void run(){
        //конкретные действия поединка
        takePosition();
        fire();
    }
}
SimpleCombat combat = new SimpleCombat();
new Thread(combat).start();
```

8.4 Объект синхронизации

Метод или блок кода, выполнение которого не должно прерываться исполнением того же самого кода, но на другом потоке, можно помечать модификатором synchronized.

Для задания синхронизации между методами нужен некий объект. Обычно таким объектом является разделяемый ресурс. Синхронизованный метод становится «владельцем» монитора объекта. Объектом синхронизации может быть ссылочный тип (объект) или класс.

8.4.1 Синхронизация доступа к состоянию объекта

Методы, обращающиеся к состоянию объекта, синхронизованы для того чтобы исключить одновременное изменение состояния.

```
public class SeniorStable {
    private int horseCount = 10;
    public void synchonized returnHorse(){
        horseCount++;
    }
    public void synchonized takeHorse(){
        horseCount--;
    }
    public int synchonized getHorseCount(){
        return horseCount;
    }
}
```

8.5 Межпотоковое взаимодействие

Следующие методы класса Object используются для реализации взаимодействия потоков между собой.

public final void wait() throws InterruptedException	Заставляет поток ожидать когда какой-то другой поток вызовет метод notify() или notifyAll() для данного объекта.
public final void notify()	Пробуждает поток, который вызвал метод wait() для этого же объекта.
public final void notifyAll()	Пробуждает все потоки, который вызвали метод wait() для этого же объекта.

8.6 Группы потоков

Потоки могут объединяться в группы с целью управления большим количеством потоков одновременно. Потоку разрешено получать информацию о своей группе.
Запрещено о других группах и о более крупных группах (являющихся надгруппами).

Если поток или группа потоков не обрабатывает исключение, то оно обрабатывается JVM.
Для установки собственного обработчика можно использовать метод `uncaughtException(Thread thread, Throwable throwable)` класса `java.lang.ThreadGroup`.
Следующий пример демонстрирует создание группы потоков.

```
public class thGroup{
    public static void main(Strings []args){
        ThreadGroup mainGr=new ThreadGroup("main group");
        Thread thread1=new Thread(mainGr, "thread1");
        ThreadGroup minorGr=new ThreadGroup(mainGr, "minor group");
    }
}
```

8.7 Исполнение по расписанию

В следующем примере сообщение выдается через несколько секунд после инициализации приложения.

```
public class TimerReminder {

    Timer timer;

    public TimerReminder(int seconds) {
        timer = new Timer();
        timer.schedule(new RemindTask(), seconds*1000);
    }

    class RemindTask extends TimerTask {
        public void run() {
            System.out.format("Time's up!%n");
            timer.cancel(); //Terminate the timer thread
        }
    }

    public static void main(String args[]) {
        System.out.format("About to schedule task.%n");
        new TimerReminder(5);
        System.out.format("Task scheduled.%n");
    }
}
```

Дополнительная литература:

- Doug Lea, Concurrent Programming in Java

8.8 Аплет «бегущая строка»

Следующий пример демонстрирует работу потока, обновляющего состояние графического интерфейса аплета через равные интервалы времени.

```
import java.applet.*;
import java.awt.*;

public class ScrollText extends Applet implements Runnable{
    Thread workThread=null;
    String str;
    int xpos;
    int ypos;
    int leftedgepos;
    int pause;

    public void paint(Graphics g){
        if (--xpos<leftedgepos)
            xpos=size().width;
        g.drawString(str,xpos,ypos);
    }
    public void init(){
        str=getParam("Greet","Hello!");
        xpos=getParam("X",100);
        ypos=getParam("Y",50);
        leftedgepos=getParam("LEFTPOS",-100);
        pause=getParam("PAUSE",100);
        setFont(new Font("Helvetica", Font.BOLD,24));
    }

    public String getParam(String p, String d){
        String s=getParameter(p);
        return s==null? d:s;
    }
    public int getParam(String p, int d){
        try {
            String s=getParameter(p);
            if (s!=null)
                d=Integer.parseInt(s);
        }catch (NumberFormatException e){ }
        return d;
    }

    public void start(){
        workThread=new Thread(this);
        workThread.start();
    }
    public void run(){
        while (true){
            repaint();
            try {
                Thread.sleep(pause);
            }catch (InterruptedException e) { }
        }
    }
    public void stop(){
```

```
        if (workThread!=null){
            workThread.stop();
        }
    }
}
```

8.9 Пример: часы реального времени

Пример показывает страничку, содержащую электронные часы. Для отсчета секунд используется отдельный поток.

```
//<applet code= "ClockOnPage.class" width =300 height=300>
//</applet>
public class ClockOnPage extends Applet{
    public void init(){
        add(new SimpleClockDisplay());
        setSize(200,75);
    }
    public void start(){}
    public void stop(){}
}

class Pulse implements Runnable{
    private ClockDisplay clock;
    public Pulse(ClockDisplay c){
        clock = c;
    }
    public void init(){
        clock.updateTime();
        new Thread(this).start();
    }
    private synchronized void tick(){
        try{
            wait(1000);
            clock.updateTime();
        }catch(InterruptedException e){}
    }
    public void run(){
        while(true){ tick();}
    }
}

interface ClockDisplay{
    public void updateTime();
}
```

```
class SimpleClockDisplay extends Label implements ClockDisplay{
    Pulse p;
    public SimpleClockDisplay(){
        setFont(new Font("Helvetica",Font.BOLD,20));
        setAlignment(CENTER);
        setForeground(Color.yellow);
        setBackground(Color.black);
        p=new Pulse(this);
        p.init();
    }
    public void updateTime(){
        Date date=new Date();
        int h=date.getHours();
        int m=date.getMinutes();
        int s=date.getSeconds();

        String hours,minutes,seconds,am_pm;

        if (h==0){
            hours=new String("12");
            am_pm=new String("am");
        }else if (h>12) {
            hours=new String(""+(h-12));
            am_pm=new String("pm");
        }else {
            hours=new String(""+h);
            am_pm=new String("am");
        }
        if (m<10) {
            minutes=new String("0"+m);
        }else{
            minutes=new String(""+m);
        }

        if (s<10) {
            seconds=new String("0"+s);
        }else{
            seconds=new String(""+s);
        }
        setText(“”+hours+":"+minutes+":"+seconds+am_pm);
   }
}
```

Домашнее задание №3 для учеников фмл N30 по спецкурсу «Java»

1. Запустите среду NetBeans, откройте проект «PlayGround»
2. Создайте свой наследник класса `java.lang.Exception`: `IgnoreException`
3. Для каждого из животных создайте метод:
   `животное.meet(другое животное) throws IgnoreException`,
   следующим образом:

   - - если животное того же вида, что и другое животное, то оно представляется;
   - - если другое животное - другого вида (например, кот встретил собаку), то наш метод бросает `IgnoreException`.

   Для сравнения видов животных вам понадобится конструкция `instanceof`

4. В методе `Playground.main()` создайте 3 животных, по одному каждого вида, виды по вашему усмотрению, - Создайте реализацию интерфейса `Runnable` (отдельный класс, давайте назовём его `Zoo`), в котором: - есть конструктор с тремя параметрами, через который вы передадите созданных животных в `Zoo`. - каждые 10 секунд 2-е из трёх животных (выбираются случайно) встречаются.
   - В методе `Playground.main()` создайте и запустите нить с классом `Zoo`, что вы написали на предыдущем шаге. - Создайте и запустите ещё одну нить с `Zoo`, работающую параллельно с первой. Измените ваш код таким образом, чтобы одно и то же животное не могло встречать кого-либо ещё одновременно в разных нитях.