Тема работы: «Обработка исключительных ситуаций»  
1. Цель работы  
Отрабатывает навык генерирования и отслеживания исключительных

ситуаций.

2. Задание

Номер варианта соответствует вашему номеру по списку.

Варианты по шифрованию:

1. Аффинная система шифрования Цезаря

2. Биграммный шифр Плейфера  
3. Двойная таблица перестановки  
4. Криптосистема Хилла

5. Магический квадрат  
6. Простая таблица перестановки  
7. Система цезаря с ключевым словам

8. Система шифрования Вижинера  
9. Система шифрования Цезаря

10.Шифр «двойной квадрат» Уинстона

11.Шифрующие таблицы Трисемуса

Необходимо в задании 1 лабораторной работы No 12 реализовать обработку исключительных ситуаций, при этом организовать:

✓ завершение работы программы, с выводом сообщения об ошибке; ✓вывода сообщения об ошибке, и организовать безопасное

продолжение работы программ;  
✓ генерирование собственного исключения.

3. Оснащение работы  
Задание по варианту, ЭВМ, среда разработки IntelliJ IDEA.

4. Основные теоретические сведения

Исключения

Исключение - это нештатная ситуация, ошибка во время выполнения программы. Самый простой пример - деление на ноль. Можно вручную отслеживать возникновение подобных ошибок, а можно воспользоваться специальным механизмом исключений, который упрощает создание больших надѐжных программ, уменьшает объѐм необходимого кода и повышает уверенность в том, что в приложении не будет необработанной ошибки.

В методе, в котором происходит ошибка, создаѐтся и передаѐтся специальный объект. Метод может либо обработать исключение самостоятельно, либо пропустить его. В любом случае исключение ловится и обрабатывается. Исключение может появиться благодаря самой системе, либо вы сами можете создать его вручную. Системные исключения возникают при неправильном использовании языка Java или запрещѐнных приѐмов доступа к системе. Ваши собственные исключения обрабатывают специфические ошибки вашей программы.

Вернѐмся к примеру с делением. Деление на нуль может предотвратить проверкой соответствующего условия. Но что делать, если знаменатель оказался нулѐм? Возможно, в контексте вашей задачи известно, как следует поступить в такой ситуации. Но, если нулевой знаменатель возник неожиданно, деление в принципе невозможно, и тогда необходимо возбудить исключение, а не продолжать исполнение программы.

Существует пять ключевых слов, используемых в исключениях: try, catch, throw, throws, finally. Порядок обработки исключений следующий.

Операторы программы, которые вы хотите отслеживать, помещаются в блок try. Если исключение произошло, то оно создаѐтся и передаѐтся дальше. Ваш код может перехватить исключение при помощи блока catch и обработать его. Системные исключения автоматически передаются самой системой. Чтобы передать исключение вручную, используется throw. Любое исключение, созданное и передаваемое внутри метода, должно быть указано в его интерфейсе ключевым словом throws. Любой код, который следует выполнить обязательно после завершения блока try, помещается в блок finally

Схематически код выглядит так:

try {

// блок кода, где отслеживаются ошибки

}

catch (тип\_исключения\_1 exceptionObject) {

// обрабатываем ошибку

}

catch (тип\_исключения\_2 exceptionObject) {

// обрабатываем ошибку

}  
finally {

// код, который нужно выполнить после завершения блока try

}

Существует специальный класс для исключений Trowable. В него входят два класса Exception и Error.

Класс Exception используется для обработки исключений вашей программой. Вы можете наследоваться от него для создания собственных типов исключений. Для распространѐнных ошибок уже существует класс RuntimeException, который может обрабатывать деление на ноль или определять ошибочную индексацию массива.

КлассErrorслужит для обработки ошибок в самом языкеJavaи на практике вам не придѐтся иметь с ним дело.

Прежде чем научиться обрабатывать исключения, нам (как и нормальному любопытному коту) хочется посмотреть, а что происходит, если

ошибку не обработать. Давайте разделим число котов в вашей квартире на ноль, хотя мы и знаем, что котов на ноль делить нельзя!

int catNumber;  
int zero;  
catNumber = 1; // у меня один кот  
zero = 0; // ноль, он и в Африке ноль  
int result = catNumber / zero;  
Я поместил код в обработчик щелчка кнопки. Когда система времени

выполнения Java обнаруживает попытку деления на ноль, она создаѐт объект исключения и передаѐт его. Да вот незадача, никто не перехватывает его, хотя это должны были сделать вы. Видя вашу бездеятельность, объект перехватывает стандартный системный обработчик Java, который отличается вредных характером. Он останавливает вашу программу и выводит сообщение об ошибке, которое можно увидеть в журнале LogCat:

Caused by: java.lang.ArithmeticException: divide by zero at ru.alexanderklimov.test.MainActivity.onClick(MainActivity .java:79)

Как видно, созданный объект исключения принадлежит к классу ArithmeticException, далее системный обработчик любезно вывел краткое описание ошибки и место возникновения.

Вряд ли пользователи вашей программы будут довольны, если вы так и оставите обработку ошибки системе. Если программа будет завершаться с такой ошибкой, то скорее всего вашу программу просто удалят. Посмотрим, как мы можем исправить ситуацию.

Поместим проблемный код в исключение.

int catNumber;

int zero;

блок try,

а в

блоке catch обработаем

try { // мониторим код  
catNumber = 1; // у меня один кот  
zero = 0; // ноль, он и в Африке ноль  
int result = catNumber / zero; Toast.makeText(this, "Не увидите это сообщение!",

Toast.LENGTH\_LONG).show();  
} catch (ArithmeticException e) {

Toast.makeText(this, "Нельзя котов делить на ноль!", Toast.LENGTH\_LONG).show();

}

Toast.makeText(this, "Жизнь продолжается", Toast.LENGTH\_LONG).show();

Теперь программа аварийно не закрывается, так как мы обрабатываем ситуацию с делением на ноль.

В данном случае мы уже знали, к какому классу принадлежит получаемая ошибка, поэтому в блоке catch сразу указали конкретный тип. Обратите внимание, что последний оператор в блоке try не срабатывает, так как ошибка происходит раньше строчкой выше. Далее выполнение передаѐтся в блок catch, далее выполняются следующие операторы в обычном порядке.

Операторы try и catch работают совместно в паре. Хотя возможны ситуации, когда catch может обрабатывать несколько вложенных операторов try.

Если вы хотите увидеть описание ошибки, то параметрeи поможет увидеть ѐго.

catch (ArithmeticException e) {  
Toast.makeText(this, e + ": Нельзя котов делить

на ноль!", Toast.LENGTH\_LONG).show(); }

По умолчанию, класс Trowable, к относится ArithmeticException возвращает строку, содержащую исключения. Но вы можете и явно указать метод e.toString.

Несколько исключений

которому описание

Фрагмент кода может содержать несколько проблемных мест. Например, кроме деления на ноль, возможна ошибка индексации массива. В таком случае вам нужно создать два или более операторов catch для каждого типа исключения. Причѐм они проверяются по порядку. Если исключение будет обнаружено у первого блока обработки, то он будет выполнен, а остальные проверки пропускаются и выполнение программы продолжается с места, который следует за блоком try/catch.

int catNumber;

int zero;

try { // мониторим код  
catNumber = 1; // у меня один кот  
zero = 1; // ноль, он и в Африке ноль  
int result = catNumber / zero;  
// Создадим массив из трѐх котов  
String[] catNames = {"Васька", "Барсик",

"Мурзик"};  
catNames[3] = "Рыжик";

Toast.makeText(this, "Не увидите это сообщение!", Toast.LENGTH\_LONG).show();

} catch (ArithmeticException e) {

Toast.makeText(this, e.toString() + ": Нельзя котов делить на ноль!", Toast.LENGTH\_LONG).show();

}

catch (ArrayIndexOutOfBoundsException e) {

Toast.makeText(this, "Ошибка: " + e.toString(), Toast.LENGTH\_LONG).show();

}

Toast.makeText(this, "Жизнь продолжается", Toast.LENGTH\_LONG).show();

В примере мы добавили массив с тремя элементами, но обращаемся к четвѐртому элементу, так как забыли, что отсчѐт у массива начинается с нуля. Если оставить значение переменной zero равным нулю, то сработает обработка первого исключения деления на ноль, и мы даже не узнаем о существовании второй ошибки. Но допустим, что в результате каких-то вычислений значение переменной стало равно единице. Тогда наше исключение ArithmeticException не сработает. Но сработает новое добавленное исключение ArrayIndexOutOfBoundsException. А дальше всѐ пойдѐт как раньше.

Тут всегда нужно помнить одну особенность. При использовании множественных операторов catch обработчики подклассов исключений должные находиться выше, чем обработчики их суперклассов. Иначе, суперкласс будет перехватывать все исключения, имея большую область перехвата. Иными словами, Exception не должен находиться выше ArithmeticException иArrayIndexOutOfBoundsException. К счастью, среда разработки сама замечает непорядок и предупреждает вас, что такой порядок не годится. Увидев такую ошибку, попробуйте перенести блок обработки исключений ниже.

Вложенные операторы try  
Операторы try могут быть вложенными. Если вложенный оператор try не

имеет своего обработчикаcatchдля определения исключения, то идѐт поиск обработчика catch у внешнего блока try и т.д. Если подходящий catch не будет найден, то исключение обработает сама система (что никуда не годится).

Оператор throw

Часть исключений может обрабатывать сама система. Но можно создать собственные исключения при помощи оператораthrow. Код выглядит так:

throw экземпляр\_Throwable

Вам нужно создать экземпляр классаThrowableили его наследников. Получить объект класса Throwable можно в оператореcatch или стандартным способом через оператор new.

Мы могли бы написать такой код для кнопки:

Cat cat;

public void onClick(View view) { if(cat == null){

throw new NullPointerException("Котик не инициализирован");

} }

Мы объявили объект класса Cat, но забыли его проинициализировать, например, в onCreate(). Теперь нажатие кнопки вызовет исключение, которое обработает система, а в логах мы можем прочитать сообщение об ошибке. Возможно, вы захотите использовать другое исключение, например, throw new UnsupportedOperationException("Котик не инициализирован");.

В любом случае мы передали обработку ошибки системе. В реальном приложении вам нужно обработать ошибку самостоятельно.

Поток выполнения останавливается непосредственно оператора throw и другие операторы не выполняются. При этом ближайший блок try/catch соответствующего исключению типа.

Перепишем пример с обработкой ошибки.

public void onClick(View view) { if (cat == null) {

после ищется

try {  
throw new NullPointerException("Кота не

существует");

} catch (NullPointerException e) {

Toast.makeText(this, e.getMessage(), Toast.LENGTH\_LONG).show();

} }

}

Мы создали новый объект класса NullPointerException. Многие классы исключений кроме стандартного конструктора по умолчанию с пустыми скобками имеют второй конструктор с строковым параметром, в котором можно разместить подходящую информацию об исключении. Получить текст из него можно через метод getMessage(), что мы и сделали в блоке catch.

Теперь программа не закроется аварийно, а будет просто выводить сообщения в всплывающих Toast.

Оператор throws

Если метод может породить исключение, которое он сам не обрабатывает, он должен задать это поведение так, чтобы вызывающий его код мог позаботиться об этом исключении. Для этого к объявлению метода

добавляется конструкция throws, которая перечисляет типы исключений (кроме исключений Error и RuntimeException и их подклассов).

Общая форма объявления метода с оператором throws:

тип имя\_метода(список\_параметров) список\_исключений {

// код внутри метода

}

throws

В фрагменте список\_исключений можно указать список исключений через запятую.

Создадим метод, который может породить исключение, но не обрабатывает его. А в щелчке кнопки вызовем его.

// Метод без обработки исключения

public void createCat(){

Toast.makeText(this, "Вы создали котѐнка", Toast.LENGTH\_LONG).show();

throw new NullPointerException("Кота не существует");

}

// Щелчок кнопки

public void onClick(View v) {

createCat();

}

Если вы запустите пример, то получите ошибку. Исправим код.

// Без изменений

public void createCat() throws NullPointerException {

Toast.makeText(this, "Вы создали котѐнка", Toast.LENGTH\_LONG).show();

throw new NullPointerException("Кота не существует");

}

// Щелчок кнопки

public void onClick(View v) {

try {

createCat();

} catch (NullPointerException e) {  
// TODO: handle exception Toast.makeText(this, e.getMessage(),

Toast.LENGTH\_LONG).show(); }

}

Мы поместили вызов метода в блок try и вызвали блок catch с нужным типом исключения. Теперь ошибки не будет.

Оператор finally

Когда исключение передано, выполнение метода направляется по нелинейному пути. Это может стать источником проблем. Например, при входе метод открывает файл и закрывает при выходе. Чтобы закрытие файла не было пропущено из-за обработки исключения, был предложен механизм finally.

Ключевое слово finally создаѐт блок кода, который будет выполнен после завершения блока try/catch, но перед кодом, следующим за ним. Блок будет выполнен, независимо от того, передано исключение или нет. Операторfinallyне обязателен, однако каждый операторtryтребует наличия либо catch, либо finally.

Встроенные исключения Java

Существуют несколько готовых системных исключений. Большинство из них являются подклассами типа RuntimeException и их не нужно включать в список throws. Вот небольшой список непроверяемых исключений.

✓ ArithmeticException - арифметическая ошибка, например, деление на нуль

* ✓ ▪ArrayIndexOutOfBoundsException - выход индекса за массива
* ✓ ▪ArrayStoreException - присваивание элементу несовместимого типа
* ✓ ▪ClassCastException - неверное приведение
* ✓ ▪EnumConstantNotPresentException - попытка   
  неопределѐнного значения перечисления
* ✓ ▪IllegalArgumentException - неверный аргумент при вызове метода
* ✓ ▪IllegalMonitorStateException - неверная операция мониторинга
* ✓ ▪IllegalStateException - некорректное состояние приложения
* ✓ ▪IllegalThreadStateException - запрашиваемая операция несовместима   
  с текущим потоком  
  ✓IndexOutofBoundsException - тип индекса вышел за допустимые   
  пределы
* ✓ ▪NegativeArraySizeException - создан массив отрицательного размера
* ✓ ▪NullPointerException - неверное использование пустой ссылки
* ✓ ▪NumberFormatException - неверное преобразование числовой формат
* ✓ ▪SecurityException - попытка нарушения безопасности
* ✓ ▪StringIndexOutOfBounds - попытка использования   
  пределами строки
* ✓ ▪TypeNotPresentException - тип не найден

строки в индекса за

массива использования

границу объекта

✓ UnsupportedOperationException - обнаружена неподдерживаемая операция

Список проверяемых системных исключений, которые можно включать в список throws.

* ✓ ▪ClassNotFoundException - класс не найден
* ✓ ▪CloneNotSupportedException - попытка клонировать объект,   
  который не реализует интерфейс Cloneable
* ✓ ▪IllegalAccessException - запрещен доступ к классу
* ✓ ▪InstantiationException - попытка создать объект абстрактного класса   
  или интерфейса
* ✓ ▪InterruptedException - поток прерван другим потоком
* ✓ ▪NoSuchFieldException - запрашиваемое поле не существует
* ✓ ▪NoSuchMethodException - запрашиваемый метод не существует
* ✓ ▪ReflectiveOperationException - исключение, связанное с рефлексией   
  Создание собственных классов исключений   
  Система не может предусмотреть все исключения, иногда вам придѐтся создать собственный тип исключения для вашего приложения. Вам нужно наследоваться от Exception (напомню, что этот класс наследуется от Trowable) и переопределить нужные методы классаThrowable. Либо вы можете унаследоваться от уже существующего типа, который наиболее близок по логике с вашим исключением.
* ✓ ▪final void addSuppressed(Throwable exception) - добавляет исключение в список подавляемых исключений (JDK 7)
* ✓ ▪Throwable fillInStackTrace() - возвращает объект класса Throwable, содержащий полную трассировку стека.   
  ✓Throwable getCause() - возвращает исключение, лежащее под текущим исключение или null
* ✓ ▪String getLocalizedMessage() - возвращает локализованное описание исключения
* ✓ ▪String getMessage() - возвращает описание исключения
* ✓ ▪StackTraceElement[] getStackTrace() - возвращает массив, содержащий трассировку стека и состояний из элементов   
  классаStackTraceElement
* ✓ ▪final Throwable[] getSuppressed() - получает подавленные   
  исключения (JDK 7)
* ✓ ▪Throwable initCause(Throwable exception) - ассоциирует исключение   
  с вызывающим исключением. Возвращает ссылку на исключение.
* ✓ ▪void printStackTrace() - отображает трассировку стека
* ✓ ▪void printStackTrace(PrintStream stream) - посылает трассировку   
  стека в заданный поток
* ✓ ▪void printStackTrace(PrintWriter stream) - посылает трассировку   
  стека в заданный поток

✓ void setStackTrace(StackTraceElement elements[]) - устанавливает трассировку стека для элементов (для специализированных приложений)

✓String toString() - возвращает объект классаString, содержащий описание исключения.

Самый простой способ - создать класс с конструктором по умолчанию.

// Если этот код работает, его написал Александр Климов,

// а если нет, то не знаю, кто его писал. package ru.alexanderklimov.exception;

import android.os.Bundle;  
import android.support.v7.app.AppCompatActivity; import android.view.View;

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main); }

public void testMethod() throws HungryCatException{

System.out.println("Возбуждаем HungryCatException из метода testMethod()");

throw new HungryCatException(); // конструктор по умолчанию

перехвачено");

}

}

}

public void onClick(View view) {

try {

testMethod();

} catch (HungryCatException e) {

e.printStackTrace(); System.out.println("Наше

исключение

class HungryCatException extends Exception{

} }

Мы  
методеtestMethod()его возбуждаем, а по нажатию кнопки вызываем этот метод. В результате наше исключение сработает.

создали собственный класс HungryCatException, в Создать класс исключения с конструктором, который получает аргумент-

строку, также просто.

// Если этот код работает, его написал Александр Климов,

// а если нет, то не знаю, кто его писал. package ru.alexanderklimov.exception;

import android.os.Bundle;  
import android.support.v7.app.AppCompatActivity; import android.view.View;

public class MainActivity extends AppCompatActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main); }

public void testMethod() HungryCatException {

System.out.println("Возбуждаем HungryCatException из метода testMethod()");

throw new HungryCatException(); конструктор по умолчанию

}

public void testMethod2() HungryCatException {

System.out.println("Возбуждаем HungryCatException из метода testMethod2()");

throws

//

throws

throw new HungryCatException("Создано во втором методе");

}

public void onClick(View view) {

try {

testMethod();

} catch (HungryCatException e) {

e.printStackTrace(); System.out.println("Наше

перехвачено");

}

try {

testMethod2();

} catch (HungryCatException e) { e.printStackTrace();

} }

исключение

class HungryCatException extends Exception {

HungryCatException() {

}

HungryCatException(String msg) {

super(msg);

} }

}

Теперь класс содержит два конструктора. Во втором конструкторе используется конструктор родительского класса с аргументомString, вызываемый ключевым словом super.

Перехват произвольных исключений

Можно создать универсальный обработчик, перехватывающий любые типы исключения. Осуществляется это перехватом базового класса всех исключений Exception:

cacth(Exception e) {  
Log.w("Log", "Перехвачено исключение");

}

Подобная конструкция не упустит ни одного исключения, поэтому еѐ следует размещать в самом конце списка обработчиков, во избежание блокировки следующих за ней обработчиков исключений.

Основные правила обработки исключений

Используйте исключения для того, чтобы:  
✓ обработать ошибку на текущем уровне (избегайте перехватывать

исключения, если не знаете, как с ними поступить)

✓исправить проблему и снова вызвать метод, возбудивший исключение

* ✓ ▪предпринять все необходимые действия и продолжить выполнение без повторного вызова действия
* ✓ ▪попытаться найти альтернативный результат вместо того, который должен был бы произвести вызванный метод
* ✓ ▪сделать все возможное в текущем контексте и заново возбудить это же исключение, перенаправив его на более высокий уровень
* ✓ ▪сделать все, что можно в текущем контексте, и возбудить новое исключение, перенаправив его на более высокий уровень
* ✓ ▪завершить работу программы
* ✓ ▪упростить программу (если используемая схема обработки   
  исключений делает все только сложнее, значит, она никуда не   
  годится)
* ✓ ▪добавить вашей библиотеке и программе безопасности   
  5. Порядок выполнения работы   
  6. Форма отчета о работе   
  Лабораторная работа No \_\_\_ Номер учебной группы \_\_\_\_\_\_\_\_\_\_\_\_   
  Фамилия, инициалы учащегося \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дата выполнения работы \_\_\_\_\_\_\_\_\_\_\_\_\_  
  Тема работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Цель работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Оснащение работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Результат выполнения работы: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
  7. Контрольные вопросы и задания   
  1. Чтотакоеисключительнаяситуация?  
  2. Каких видов существуют исключительные ситуации? 3. Как отловить исключительную ситуацию?  
  4. Как сгенерировать исключительную ситуацию?

1. построить объектную модель;  
2. построить алгоритм решения данной задачи;  
3. реализовать приложение;  
4. провести тестирование полученного приложения.