Обработка ошибок, исключения, отладка

Алексей Владыкин

29 сентября 2014

1. Подходы к обработке ошибок
2. Исключения
3. Java Logging API
4. Отладка и диагностика


© 17/03/04 Interwizard2001
Windows 95
An error occured whilst trying to load the application, "bluescreen.exe"
Error Error
An error occured whilst trying to load the previous error.
OK
A fatal e
VXB UMM (82)
88018E35
* Press an
* Press CT
. You will
lose any
* Buy a Mac you fucktart.
Press any key to continue _
Hey, it looks like you're having an error!

- Ошибки игнорируются
- При ошибке программа аварийно завершается
- При ошибке функция вместо результата возвращает специальное значение
- Функция возвращает признак ошибки отдельным выходным параметром
- Функция возвращает признак ошибки в глобальной переменной
- При ошибке функция бросает исключение

Что такое «исключение»

- Исключение (exception) — событие, возникающее в процессе работы программы и прерывающее её нормальное исполнение

Примеры:

- `java.lang.NullPointerException`
- `java.lang.ClassCastException`
- `java.lang.OutOfMemoryError`
- `java.io.IOException`

Пример исключения

```
java.lang.NullPointerException
    at ru.compscicenter.java2014.Test.baz(Test.java:19)
    at ru.compscicenter.java2014.Test.bar(Test.java:14)
    at ru.compscicenter.java2014.Test.foo(Test.java:10)
    at ru.compscicenter.java2014.Test.main(Test.java:6)
```

java.lang.Throwable

- Исключение в Java — полноценный объект
- Все исключения в Java наследуются от класса Throwable

- `String getMessage()`
- `StackTraceElement[] getStackTrace()`
- `void printStackTrace()`
- `Throwable getCause()`
- `Throwable[] getSuppressed()`

Классификация исключений

- Исключительные ситуации в JVM
  `java.lang.Error`
- Исключительные ситуации в пользовательском коде
  - Проверяемые (checked)
    `java.lang.Exception`
  - Непроверяемые (unchecked)
    `java.lang.RuntimeException`

Выброс исключения

```
public class CalculatorImpl implements Calculator {
    @Override
    public double calculate(String expr) {
        if (expr == null) {
            throw new NullPointerException("expr is null");
        }
        // or using utility method:
        // Objects.requireNonNull(expr, "expr is null");
        // ...
    }
}
```

- Оператор `throw` прерывает нормальное исполнение программы и запускает поиск обработчика исключения
- Если исключение проверяемое, метод должен содержать его в списке `throws`

Объявление нового типа исключения

```
public class CalculatorException extends RuntimeException {

    public CalculatorException(String message) {
        super(message);
    }

    public CalculatorException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

Перехват исключения: try-catch

```
System.out.print("Please enter expression: ");
for (;;) {
    String expr = readUserInput();
    if (expr == null || "exit".equalsIgnoreCase(expr)) {
        break;
    }
    try {
        double result = calculator.calculate(expr);
        System.out.println("Result: " + result);
    } catch (CalculatorException e) {
        System.out.print("Bad expression, try again: ");
    }
}
```

Перехват нескольких исключений

```
try {
    // ...
} catch (FirstException e) {
    e.printStackTrace();
} catch (SecondException e) {
    e.printStackTrace();
}

// since Java 7 can be replaced with:
try {
    // ...
} catch (FirstException | SecondException e) {
    e.printStackTrace();
}
```

Обработка исключения

- Игнорирование (пустой `catch`)
- Запись в лог
- Проброс дальше того же или нового исключения
- Содержательная обработка (например, повтор операции)

Исключения и освобождение ресурсов

```
InputStream is = new FileInputStream("a.txt");
try {
    readFromInputStream(is);
} finally {
    is.close();
}
```

- Блок `finally` будет выполнен в любом случае
- В нем обычно освобождают использованные ресурсы

try с ресурсами

```
try (InputStream is =
        new FileInputStream("a.txt")) {
    readFromInputSteam(is);
}
```

- Метод `close()` будет вызван автоматически, как в `finally`
- Можно перечислить сразу несколько ресурсов
- Ресурсы должны реализовать интерфейс `java.lang.AutoCloseable`
- Добавлен в Java 7

Гарантии безопасности

- Гарантии отсутствия исключений
- Сильные гарантии
- Слабые гарантии
- Гарантия отсутствия утечек
- Никаких гарантий


STOP!
ILLEGAL LOGGING

- Пакет `java.util.logging`
- Центральная сущность — `java.util.logging.Logger`
- Логгеры образуют иерархию

```
Logger logger = Logger.getLogger(
        "ru.compscicenter.java2014.logging");

logger.info("I'm logging!");
```

Почему не `System.out.println`

- Логгеры включаются/отключаются и настраиваются без перекомпиляции
- Возможно логирование в консоль, в файл, по сети...
- Возможны разные форматы лога: текст, XML, ...

- Уровни логирования:
  SEVERE, WARNING, INFO, CONFIG, FINE, FINER, FINEST

```
logger.setLevel(Level.INFO);

logger.fine("I'm still logging");

logger.log(Level.WARNING,
        "Houston, we have a problem!");
```

java.util.logging.Handler

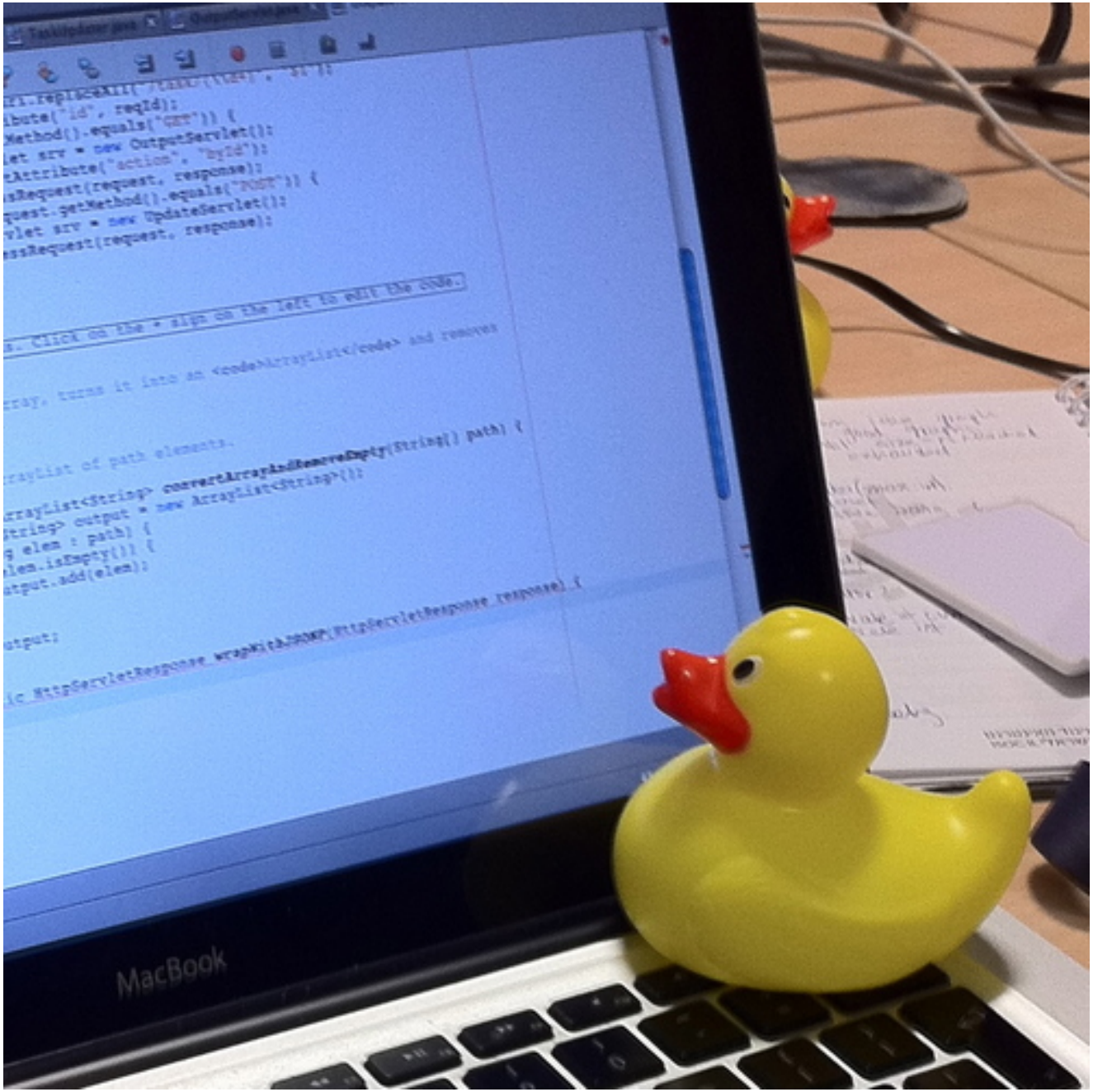
- Обработчик сообщения
  Определяет, куда будет записано сообщение

- `java.util.logging.ConsoleHandler`
- `java.util.logging.FileHandler`

java.util.logging.Formatter

- Оформитель сообщения
  Определяет формат вывода

- `java.util.logging.SimpleFormatter`
- `java.util.logging.XMLFormatter`


MacBook

Демо

- Отладка приложения, запущенного в IDE
- Отладка приложения, запущенного отдельно
- Visual VM
- Mission Control

Что сегодня узнали

- Какие есть подходы к обработке ошибок
- Как пользоваться исключениями
- Как пользоваться стандартным API для логирования
- Какие инструменты могут помочь в диагностике проблем