Основы программирования на Java

Исключения

Программа

- Классы и объекты в Java: основные сведения
- Управление памятью для ссылочных типов
- Реализация наследования в Java
- Пакеты как механизм реализации пространств имен в Java
- Абстрактные классы и интерфейсы
- Настраиваемые типы
- **Обработка исключений**
- Многопоточное программирование

Обработка исключений

- Исключение
  - Объект, создаваемый специальным образом, описывающий исключительную ситуацию
    - Процесс создания такого объекта обычно называют «выбрасыванием», или генерацией, исключения
  - В целом модель генерации и обработки исключений похожа на C++
    - Важное отличие: исключения всегда создаются в динамической памяти (управляются как и все другие объекты Java)

Demo: UncaughtExceptionSample

Обработка исключений

- Что будет, если не обработать исключение, сгенерированное программой?

```
public class Main {
    static int division( int a, int b ) {
        return a/b;
    }
    public static void main( String args[] ) {
        int result = division( 10, 0 );
    }
}
```

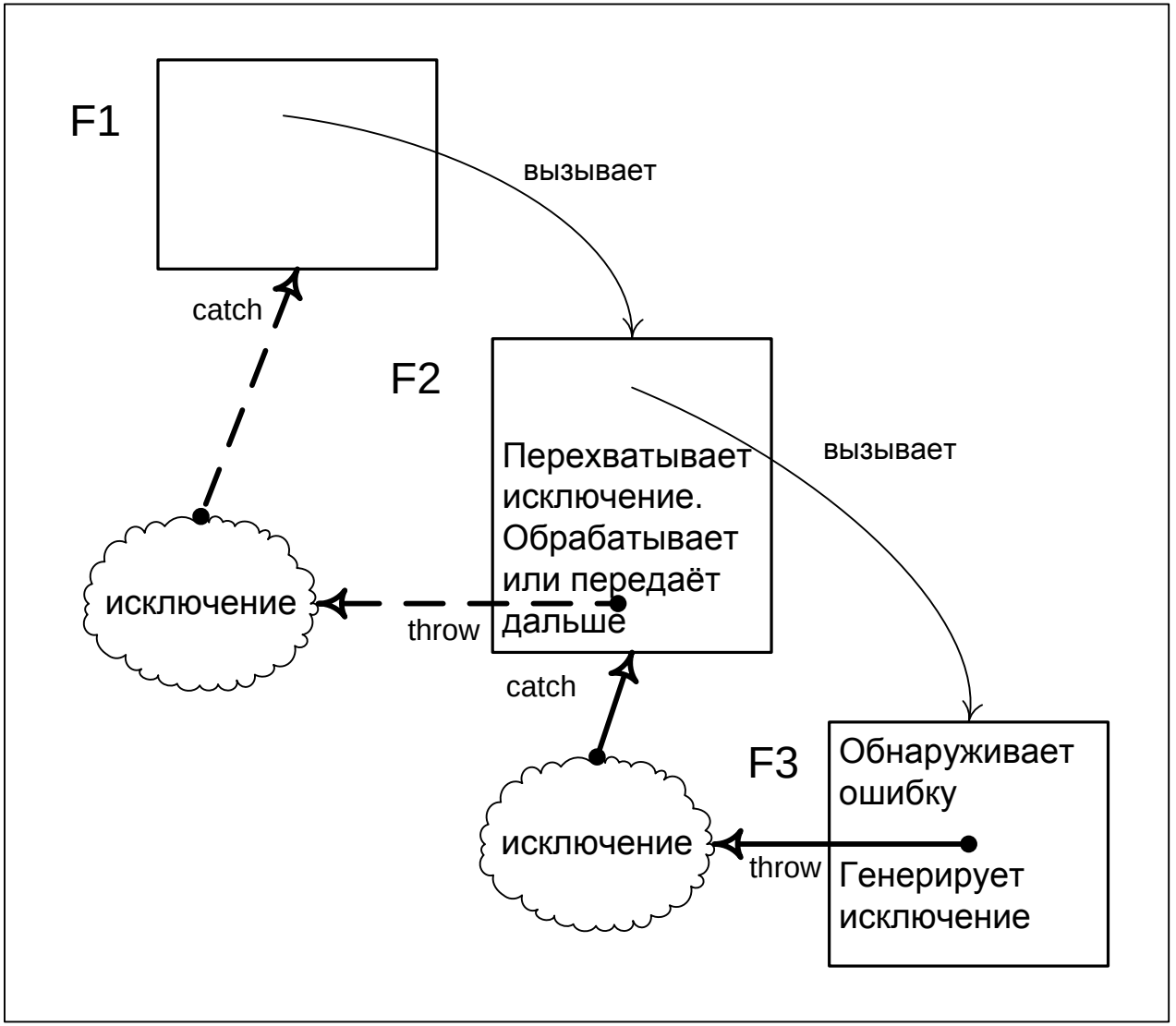
Demo: UncaughtExceptionSample

Обработка исключений

- Что будет, если не обработать исключение, сгенерированное программой?
  - > Исключение захватывается обработчиком, заданным исполнительной системой Java по умолчанию
  - > На экране отображается трасса стека

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at uncaughtexceptionsample.Main.division(Main.java:15)
        at uncaughtexceptionsample.Main.main(Main.java:18)
```

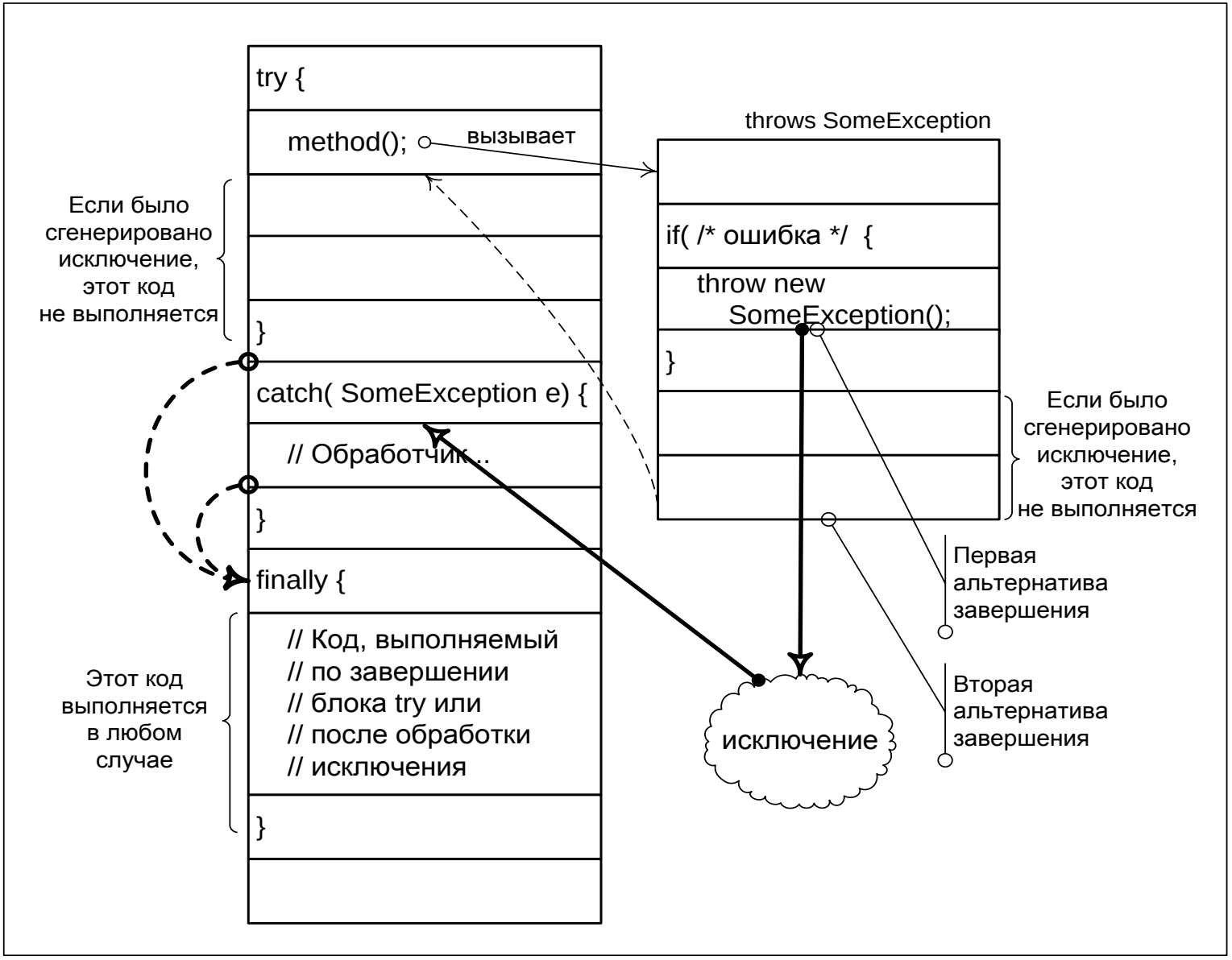
Обработка исключений: общая схема вызовов



Обработка исключений

- Общая схема вызовов: комментарий
  - Любое исключение имеет тип, производный от класса Throwable
  - В процессе обработки можно получить информацию о состоянии стека приложения.
  - «По следам стека» можно установить проблемный метод и причину ошибки

Обработка исключений: try-catch-finally



Demo: ExceptionGeneralCaseSample

Обработка исключений

- Простой пример для ArithmeticException
  - Перехватываем и обрабатываем исключение, возникающее при делении на 0

Demo: ExceptionGeneralCaseSample

Обработка исключений

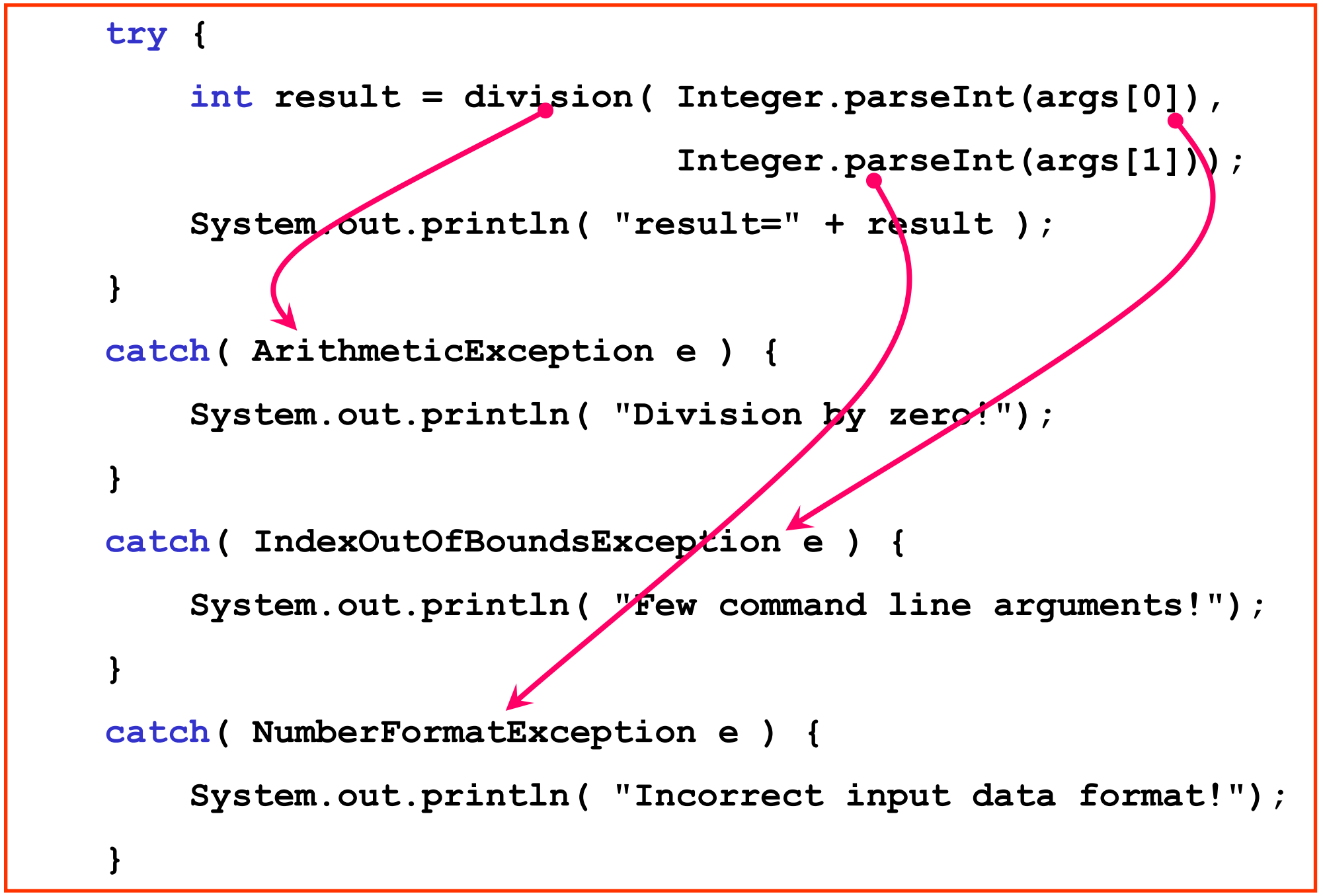
```
try {
    int result = division( Integer.parseInt(args[0]),
                           Integer.parseInt(args[1]));
    System.out.println( "result=" + result );
}
catch( ArithmeticException e ) {
    System.out.println( "Division by zero!");
}
finally {
    System.out.println( "Safe division completed");
}
```

Demo: ExceptionGeneralCaseSample

Обработка исключений

- Простой пример для ArithmeticException
  - > А если забудем задать аргументы командной строки?
  - > Или эти аргументы не смогут быть интерпретированы как целые числа

  - > ***Вывод:*** необходимо предусмотреть обработку и других исключительных ситуаций
  - > Блоков catch может быть несколько


try {
int result = division(Integer.parseInt(args[0]),
Integer.parseInt(args[1]));
System.out.println("result=" + result);
}
catch(ArithmeticException e) {
System.out.println("Division by zero!");
}
catch(IndexOutOfBoundsException e) {
System.out.println("Few command line arguments!");
}
catch(NumberFormatException e) {
System.out.println("Incorrect input data format!");
}

Demo: ExceptionGeneralCaseSample

Обработка исключений

- Каскад обработчиков
  - Порядок записи обработчиков имеет значение
    - Нужно располагать обработчики более частных типов исключений раньше обработчиков более общих типов исключений
  - Блок **`catch`**`(...)`
    - `Этому блоку передается управление, если ни один из обработчиков выше по тексту не подходит`

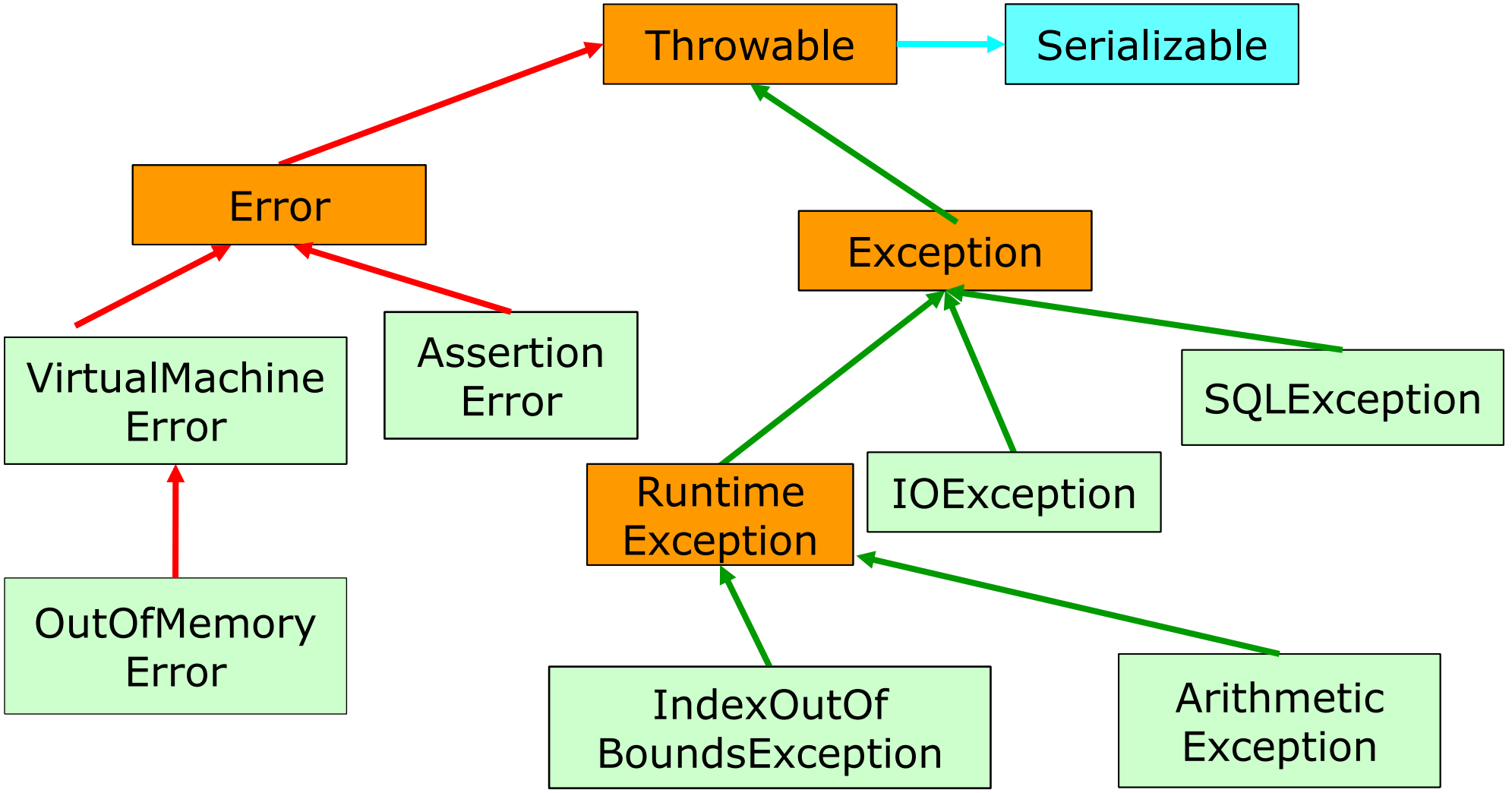
Обработка исключений

- Виды исключений
  - Исключения, выбрасываемые исполнительной системой
    - Примеры: **ArithmeticException, IndexOutOfBoundsException, NumberFormatException,…**
    - Другое название – ***неконтролируемые исключения***
      - Так как компилятор не проверяет, выбрасывает или обрабатывает метод эти исключения
    - Производные от **java.lang.RuntimeException**

Обработка исключений

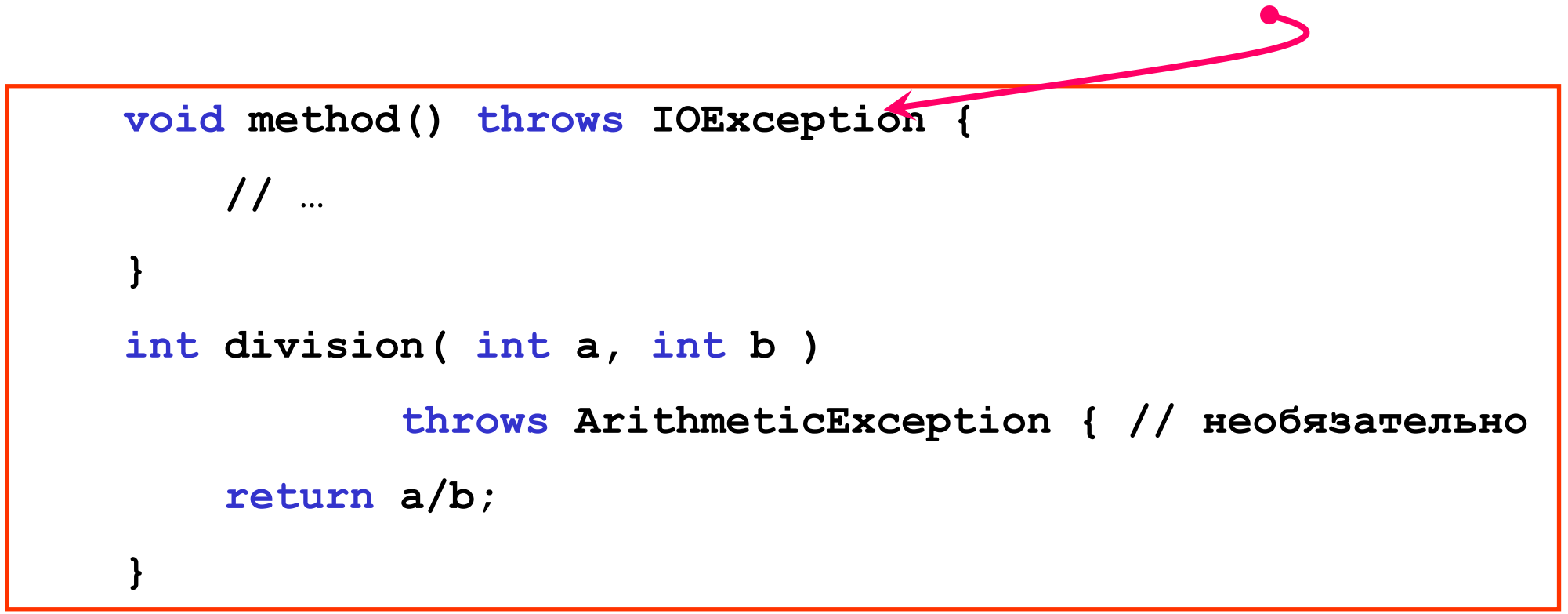
- Виды исключений
  - Контролируемые исключения
    - Примеры: **FileNotFoundException, IOException,…**
    - Обязательны для описания при определении метода, который может выбрасывать эти исключения, поэтому контролируются компилятором
    - Производные от **java.lang.Exception**
  - Ошибки
    - Примеры: **VirtualMachineError, OutOfMemoryError,…**
    - Производные от **java.lang.Error**

Иерархия типов исключений



Обработка исключений

- Спецификация исключений
  - > Описание выбрасываемых исключений при определении метода
  - > ***для контролируемых исключений – обязательна!***



```
void method() throws IOException {
    // …
}
int division( int a, int b )
            throws ArithmeticException { // необязательно
    return a/b;
}
```

Обработка исключений

- Если метод выбрасывает контролируемое исключение, он должен
  - > Либо содержать обработку исключения (catch)
  - > Либо содержать спецификатор throws в заголовке метода

- ***Код, не удовлетворяющий этому правилу, не компилируется!***

Обработка исключений

- Определение собственных классов исключений
  - > В большинстве случаев достигается определением класса, производного от ***Exception***
  - > ***наследуются все методы из Throwable***

```
class SpecificException extends Exception {

    // Очень часто ничего не придется реализовывать,

    // достаточно того, что есть в Throwable

}
```

Обработка исключений

- Методы, унаследованные из **`Throwable`**

`Throwable fillInStackTrace()`	Возвращает Throwable-объект, содержащий полную трассу стека. Может быть выброшен повторно
`String getLocalizedMessage()`	Возвращает локализованное описание исключения.
`String getMessage()`	Возвращает описание исключения.
`void printStackTrace()`	Отображает трассу стека. Имеются версии этого метода для работы с потоками
`String toString()`	Возвращает String-объект, содержащий описание исключения

Demo: UserExceptionSample

Обработка исключений

- Определение собственных классов исключений
  - > Можно определить дополнительные методы и/или переопределить методы из Throwable

```
class TooHeavyBirdException extends Exception {
    String explain;
    public MyException( String ex ) { explain = ex; }
    public String toString() {
        return super.toString() + "[" + explain + "]";
    }
}
```

Demo: UserExceptionSample

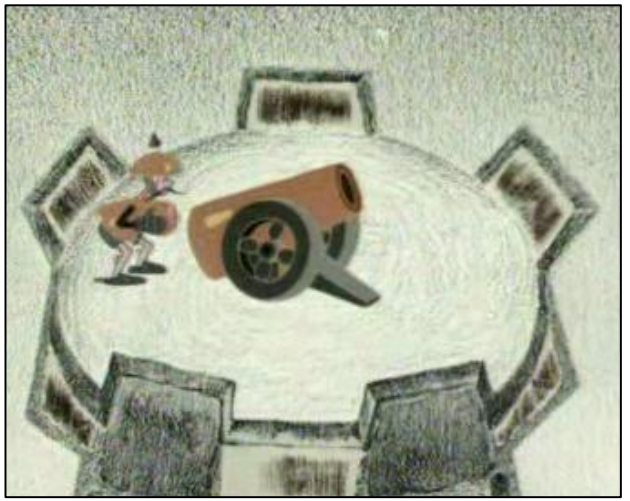
Обработка исключений

- Генерация исключения

```
class SomeClass {
    static void throwMethod() throws TooHeavyBirdException {
        throw new TooHeavyBirdException("Веревка порвалась");
    }
}
```

Обработка исключений

- **Вредные советы**
  - Используйте блок catch(…), ведь это быстро и эффективно позволит обработать любое исключение
  - Чтобы не возиться с каскадами обработчиков, используйте преимущественно catch(Exception e)
  - Используйте механизм обработки исключений не только для обработки ошибок, но и для реализации свободных от ограничений нелокальных переходов



Q&A