Лекция 10. Исключения.

- Введение
- Исключения и ошибки
- Проверяемые и непроверяемые исключения
- Блок try-catch-finally
- Оператор throw
- Зарезервированное слово throws
- Некоторые типы исключений

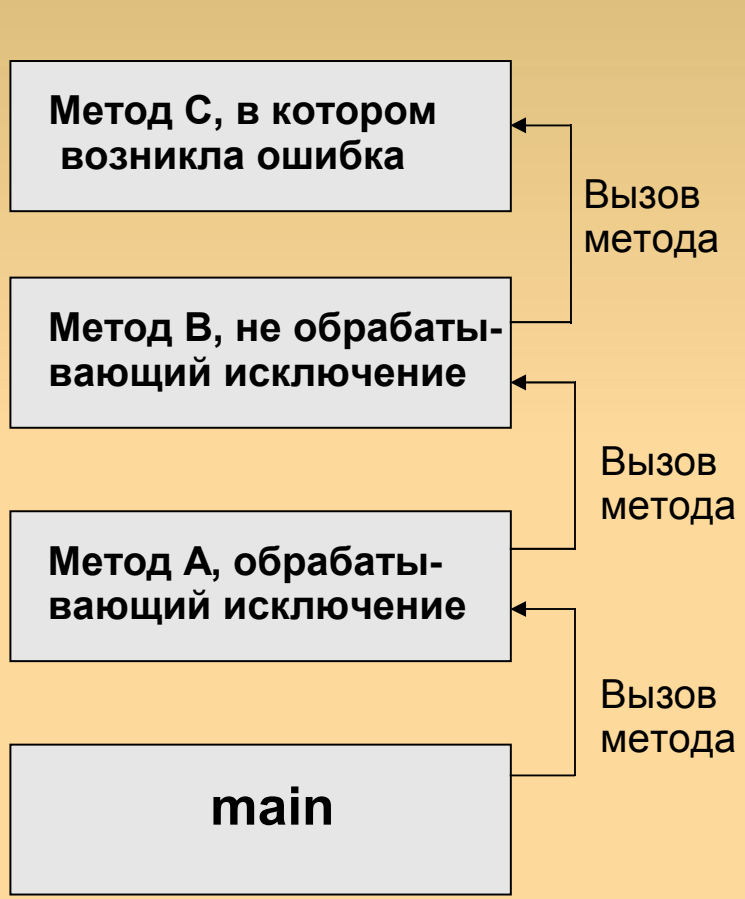
Что должно произойти при исполнении этой программы?

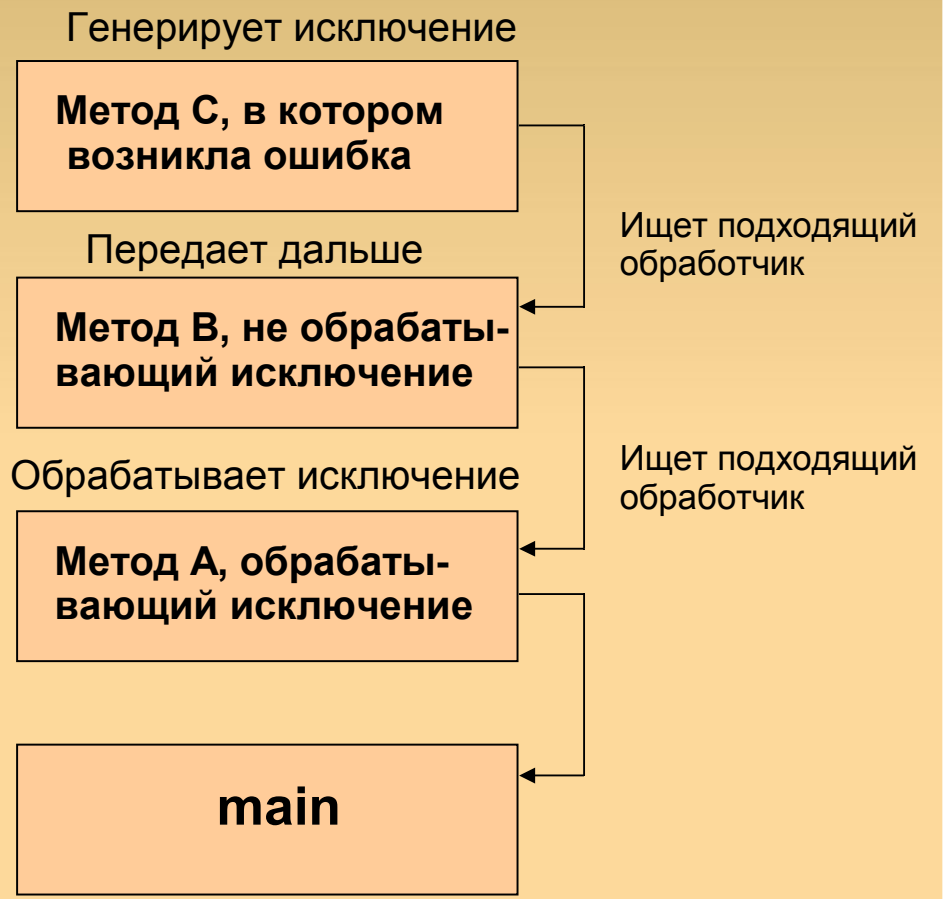
```
class SimpleMistake {
    public static void main(String args[]) {
          System.out.println(1/0);
    }
}
```

Что такое Исключение?

- Исключение в Java — это объект, который описывает исключительное состояние, возникшее в каком-либо участке программного кода.
- Все исключения в Java реализуют интерфейс Throwable.
- Этот объект пересылается в метод, обрабатывающий данный тип исключительной ситуации.
- По «следам» стека программы можно найти данный метод – и причину ошибки.

Схема возникновения и обработки исключений





Что исключение поможет найти?

- 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.

- В разработке можно использовать средства :
  - отладка (debug)
  - точка остановка выполнения программы (breakpoint)

Возникновение исключения

Совершаем преднамеренную ошибку – делим на ноль.

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

Обработка исключения

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");
      }
  }
}
```

Как действует связка try-catch

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

try {

} catch (...) {

}

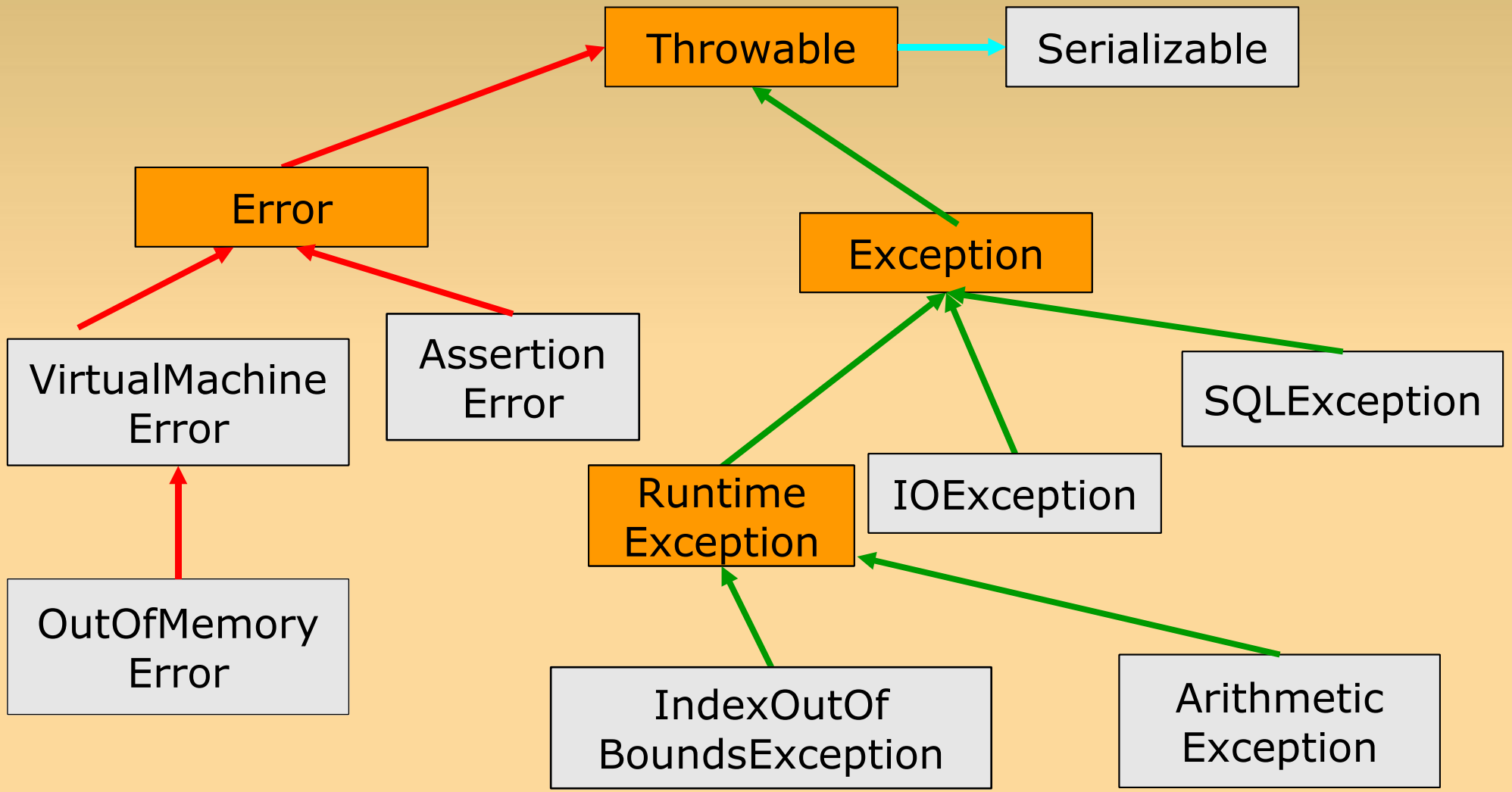
Виды исключений

- Проверяемое (производное от Exception)
  - FileNotFoundException, IOException, …
  - После такого исключения обычно возможно восстановление состояния программы.
  - Обязательны для описания при определении метода.
- Ошибка
  - Класс java.lang.Error и его наследники.
- Исключение времени исполнения (производное от RuntimeException)
  - ArithmeticExcption, ClassCastException, NegativeArraySizeException

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 виртуальной машины Java.
AWTError	Ошибка при работе графического интерфейса.

Иерархия Throwable объектов



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.

Каскад обработчиков

Иногда одного обработчика недостаточно – создаем несколько, на разные типы исключений.

```
class MultiCatch {
  public static void main (String args[]) {
      try {
            riskyMethod();
      }
      catch (ArithmeticException e) {
            tryToHandleArithmetic();
      }
      catch(ArrayIndexOutOfBoundsException e) {
             tryToHandleIndex();
      }
  }
}
```

Вложенные блоки

- Также можно вкладывать один блок try-catch в другой.
- Число вложений ограничено реализацией JVM.

```
class LevelNest{
   public static void main(String args[]) {
     try {
        //может бросить арифметическое исключение
        doSomePreparation();
       //тоже может бросить, но наружу
       //оно передано не будет - см. следующий слайд.
        doCalculation ();
     } catch (ArithmeticException e) {
          tryToHandle();
     }
   }
}
```

Вложенные блоки (продолжение)

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

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

Явно брошенное 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;
   }
 }
```

Описание исключений

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

Обязательно указываются все проверяемые

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

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();
}
```

Пользовательские классы-исключения

Создаем свой класс исключений на основе класса Exception.

```
class TooHeavyBirdException extends Exception {
      private int weight;
      private String message;
      TooHeavyBirdException (int weight, String
                  message) {
            this. weight = weight;
             this. message = message;
   }
}
```

Пользовательские классы-исключения (продолжение)

Используется обычным способом.
Обычно создавать свои исключения не требуется.

```
try {//какие-то действия
    if (condition){//бросаем наше исключение
    throw new TooHeavyBirdException (10, “Веревка не
  выдержала птицу");
}catch (TooHeavyBirdException e) {
    showModalDialog(e.getMessage());
}
```

Оборачивание исключений

- Начиная с Java 1.4, у Throwable новый конструктор: Throwable(Throwable cause)
- Проверяемые исключения можно помещать внутрь непроверяемого и перебрасывать выше по стеку вызовов
- Таким образом, исключение можно обработать там, гда позволяет логика программы без лишних throws в промежуточных участках кода.

Вредные советы

- Закрыть все опасные участки пустыми обработчиками (catch(...){ }).
  - > Быстро и эффективно.
- Использовать везде catch(Exception e){...} и throws Exception.
  - > Зачем размениваться на частные случаи и возиться с каскадами обработчиков?

Верно ли что...

- Все исключения в Java наследуют Exception?
- В одном блоке catch можно обрабатывать различные исключения?
- У каждого блока try должен быть свой блок catch?
- Неконтролируемые исключения так называются, потому что могут возникнуть где угодно и их невозможно предотвратить?
- Exception – это интерфейс?
- Блок finally выполняется в любом случае?
- Исключение можно возбудить преднамеренно?
- Исключения нельзя передавать в другие потоки?