Обработка ошибок и исключения

Обнаружение и обработка ошибок

- Плохо написанная программа не должна запускаться.
- Ошибки должны обнаруживаться во время компиляции.
- Остальные проблемы приходится решать во время работы программы, с помощью механизма, который позволяет источнику ошибки передать необходимую информацию о ней получателю - а последний справляется с возникшими трудностями.
- Механизм исключений упрощает создание надежных программ и уменьшает объем кода.

Основные исключения

- Исключительная ситуация - это проблема, из-за которой нормальное продолжение работы программы невозможно.
- Важно отличать исключительную ситуацию от обычных ошибок. Это означает, что в текущем контексте имеется достаточно информации о том, как ее обработать.
- В исключительной ситуации обработка ошибки в текущем контексте невозможна, так как программист не располагает необходимой информацией.

Пример

Пример с операцией деления:

10 / 0

Если знаменатель равен нулю, то это обычная ошибка.

Если знаменатель стал нулем неожиданно, то деление в принципе невозможно и нужно выбросить исключение.

Пример

При этом происходит следующее:

- Создается объект, представляющий исключение.

- Поток выполнения останавливается и извлекается ссылка на объект, представляющий исключение.

- Запускается механизм обработки исключений, который ищет подходящее место в программе для передачи исключения.

- Таким местом является обработчик исключений, который пытается решить проблему так, чтобы программа могла продолжить выполнение.

Создание исключения

● Чтобы передать информацию об ошибке на уровень выше для последующей обработки, создайте объект, представляющий передаваемую информацию, и запустите его из текущего контекста:

```
if (cat == null)
throw new NullPointerException();
```

● Исключения позволяют остановить программу и сообщить об ошибках или разобраться с проблемами и продолжить выполнение.

Создание исключения

- Объект, представляющий исключение, возвращается методом, несмотря на то, что в нем (методе) обычно предусмотрен другой тип.
- Однако, в отличие от обычного возврата из метода, при выбросе исключения мы попадаем совсем не туда, куда попали бы в нормальной ситуации.
- Можно выбросить исключение любого типа, происходящего от Throwable. Информация об ошибке содержится внутри объекта исключения или определяется по типу объекта.

Ключевых слова

- **try** – определяет блок кода, в котором может произойти исключение;
- **catch** – определяет блок кода, в котором происходит обработка исключения;
- **finally** – определяет блок кода, который является необязательным, но при его наличии выполняется в любом случае независимо от результатов выполнения блока try.
- **throw** – используется для возбуждения исключения;
- **throws** – используется в сигнатуре методов для предупреждения, о том что метод может выбросить исключение.

Перехват исключений

- Защищенная секция - часть программы, в которой могут произойти исключения. За ней следует специальный блок, отвечающий за обработку этих исключений:

```
try {
// Код, способный выбрасывать исключения.
}
```

- С исключениями программу легче читать и писать, иначе для проверки ошибок необходимо было бы добавить дополнительный код к вызову каждого метода.

Обработчики исключений

● Исключение должно быть обработано. Для этого предназначены обработчики исключений, которые размещаются в блоках *catch*:

```
try {
// Код, способный выбрасывать исключения.
}
catch(ExceptionType1 e) {
// Обработка исключения ExceptionType1
}
catch(ExceptionType2 e) {
// Обработка исключения ExceptionType2
}
```

Блок catch

- Каждое предложение catch принимает только один аргумент - исключение определенного типа.
- При возникновении исключения ищется первый из обработчиков соответствующего типа и передается управление в блок catch, после чего исключение считается обработанным.
- Внутри try несколько методов могут вызвать исключение одного типа, но обработчик будет вызван только один.

Прерывание и возобновление

- В обработке исключений есть две модели:
    - Модель прерывания предполагает наличие серьезной ошибки, после которой выполнение программы становится невозможным.
    - Модель возобновления подразумевает, что обработчик ошибок что-то сделает для исправления

ситуации.

Блок finally

- Часто встречаются ситуации, когда некоторая часть программы должна выполняться независимо от того, было исключение или нет. Для этого используется блок *finally*.
- Одно из применений: использование для освобождения занятых ресурсов.

Спецификация исключений

- Метод, выбрасывающий исключения должен содержать в своем описании конструкцию, указывающую программисту, исключения каких типов он генерирует:

void getParameters(BigInteger objectId) throws RemoteException, ObjectNotFoundException

- Метод, не вырабатывающий исключений:

void getParameters(BigInteger objectId)

Спецификация исключений

- Если метод выбрасывает исключения, то обойтись без спецификации нельзя. В противном случае компилятор выдаст ошибку.
- Исключения, которые проверяются и навязываются еще на этапе компиляции, называются контролируемыми или проверяемыми (checked).

Перехват произвольных исключений

- Универсальный обработчик, способный перехватывать исключения всех типов:

```
catch(Exception e) { … }
```

- Таким образом не будет упущено ни одного исключения, однако располагать этот обработчик нужно после всех остальных:

```
try { … }
catch(FileNotFoundException e) { … }
catch(IOException e) { … }
catch(Exception e) { … }
```

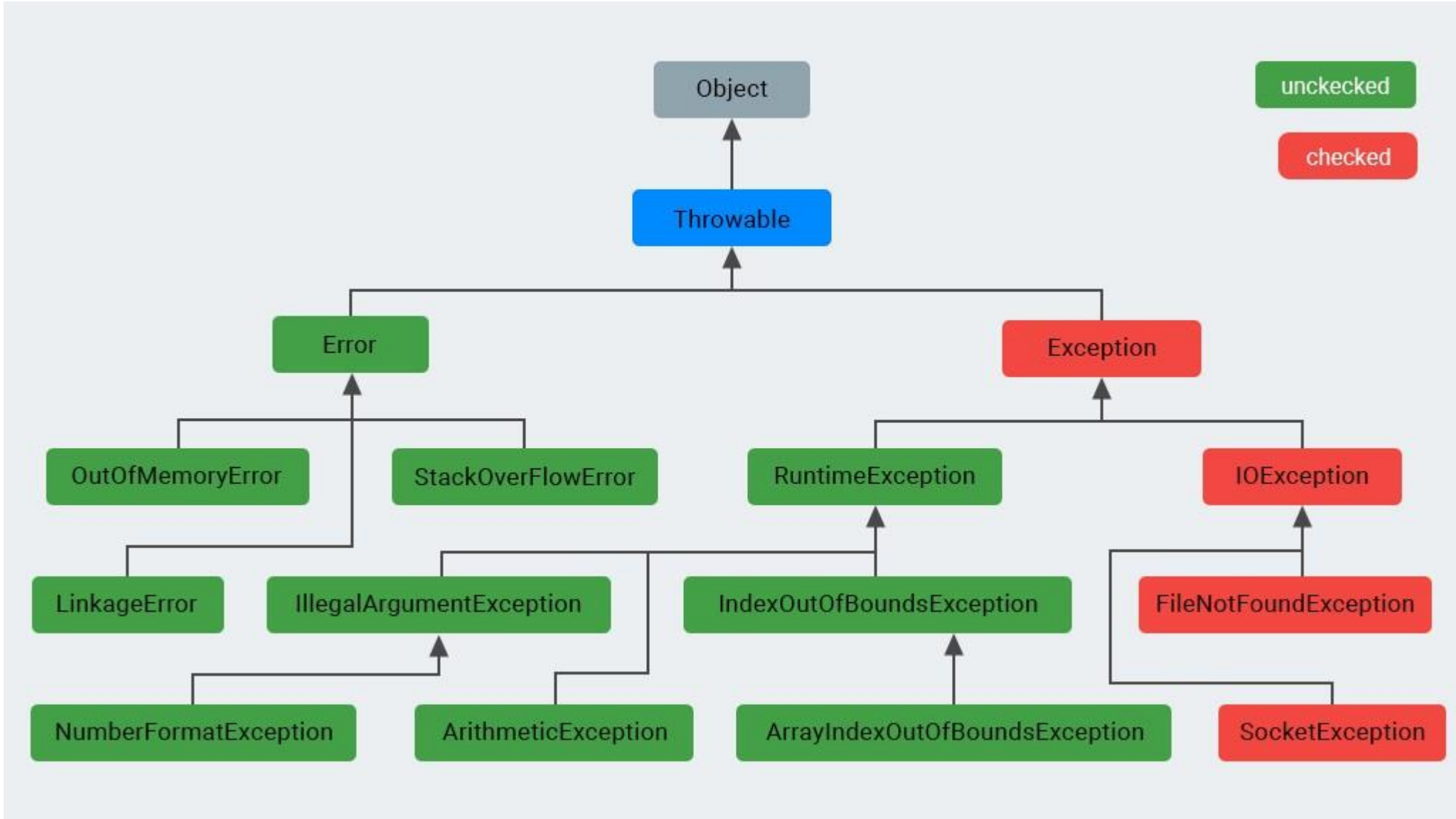
Повторный выброс исключения

● Иногда необходимо повторно выбросить перехваченное исключение, например, когда требуется регистрация сообщения об ошибке в журнале (лог-файле):

```
catch(Exception e) {
log.error(e.getMessage());
throw e;
}
```

Стандартные исключения

- Тип *Error* представляет системные ошибки и ошибки компиляции, которые обычно не перехватываются.
- Тип *Exception* может быть сгенерирован из любого метода программы.


Object
unckecked
checked
Throwable
Error
Exception
OutOfMemoryError
StackOverFlowError
RuntimeException
IOException
LinkageError
IllegalArgumentException
IndexOutOfBoundsException
FileNotFoundException
NumberFormatException
ArithmeticException
ArrayIndexOutOfBoundsException
SocketException

Throwable

Самым базовым классом для всех исключений является класс Throwable. В классе Throwable содержится код, который записывает текущий стек-трейс вызовов функций в массив.

В оператор throw можно передать только объект класса-наследника Throwable. И хотя теоретическим можно написать код вида throw new Throwable(); — так обычно никто не делает. Главная цель существования класса Throwable — единый класс-предок для всех исключений.

Error

Следующим классом исключений является класс Error — прямой наследник класса Throwable. Объекты типа Error (и его классов-наследников) создает Java-машина в случае каких-то серьезных проблем. Например, сбой в работе, нехватка памяти, и т.д.

Обычно вы как программист ничего не можете сделать в ситуации, когда в программе возникла ошибка типа Error: слишком серьезна такая ошибка. Все, что вы можете сделать — уведомить пользователя, что программа аварийно завершается или записать всю известную информацию об ошибке в лог программы.

Exception

Исключения типа Exception — это обычные ошибки, которые возникают во время работы многих методов. Цель каждого выброшенного исключения — быть захваченным тем блоком catch, который знает, что нужно сделать в этой ситуации.

Когда какой-то метод не может выполнить свою работу по какой-то причине, он сразу должен уведомить об этом вызывающий метод, выбрасывая исключение соответствующего типа.

Другими словами, если какая-то переменная оказалась равна null, метод выкинет NullPointerException, если в метод передали неверные аргументы — выкинет InvalidArgumentException, если в методе случайно было деление на ноль — ArithmeticException.

RuntimeException

- Существует группа исключений, наследующих от RuntimeException, которые выбрасываются автоматически.
- Их можно не включать в спецификацию.
- Такие исключения означают ошибки в программе и называются неконтролируемыми (unchecked).
- Обычно исключения такого типа не перехватываются (но иногда выбрасываются).

Checked exceptions

Если метод выбрасывает checked-исключение, он должен содержать тип этого исключения в своем заголовке (сигнатуре метода). Чтобы все методы, которые вызывают данный метод, знали о том, что в нем может возникнуть такое «важное исключение».

Указывать checked-исключения надо после параметров метода после ключевого слова throws (не путать со throw). Выглядит это примерно так:

```
тип метод (параметры) throws исключение
```

Examples

```
public void calculate(int n) throws Exception
{
    if (n == 0)
        throw new Exception("n равно нулю!");
}
```

```
public void calculate(int n) throws Exception, IOException
{
    if (n == 0)
        throw new Exception("n равно нулю!");
    if (n == 1)
        throw new IOException("n равно единице");
}
```

Пример. Метод выбрасывает 2 исключения

```
public void создатьМир(int n) throws ПустойМир,ОдинокийМир
{
    if (n == 0)
        throw new ПустойМир("Людей вообще нет!");
    if (n == 1)
        throw new ОдинокийМир("Слишком мало людей!");
    System.out.println("Создан прекрасный мир. Население: " + n);
}
```

Как его можно вызвать?

Не перехватываем возникающие исключения

```
public void создатьНаселенныйМир(int population)
        throws ПустойМир, ОдинокийМир
{
    создатьМир(population);
}
```

Перехватывать часть исключений

```
public void создатьНепустойМир(int population)
        throws ПустойМир
{
    try
    {
        создатьМир(population);
    }
    catch (ОдинокийМир e)
    {
        e.printStackTrace();
    }
}
```

Перехватываем все исключения

```
public void создатьЛюбойМир(int population)
{
    try
    {
        создатьМир(population);
    }
    catch(ОдинокийМир e)
    {
        e.printStackTrace();
    }
    catch(ПустойМир e)
    {
        e.printStackTrace();
    }
}
```

Обертывание исключений

Проблем от *checked-исключений* гораздо больше, чем пользы. В общем, сейчас мало кто их любит и мало кто использует.

Java-программисты предложили «заворачивать» checked-исключения внутрь RuntimeException. Другими словами, перехватывать все checked-исключения, создавать вместо них unchecked-исключения (например, RuntimeException) и выбрасывать уже их. Выглядит это все примерно так:

Пример

```
try

    {
        код, где может возникнуть checked - исключение
    }
catch(
    Exception exp)

    {
        throw new RuntimeException(exp);
    }
```

Множественный перехват исключений

```
try

    {
        //код, где может возникнуть ошибка
    }
catch(ТипИсключения1 |ТипИсключения2 |
    ТипИсключения3 имя)

    {
      // код обработки исключений
    }
```

Но нельзя написать catch (Exception | RuntimeException e), т.к. класс RuntimeException унаследован от Exception.

Собственные исключения

```
class ИмяКласса extends Exception {

 }
```

Try-with-resources

```
try(
    InputStream is = new FileInputStream("c:/file.txt");
    OutputStream os = new FileOutputStream("c:/output.txt")
)
    {
        is.read(…);
        os.write(…);
    }
```

Рекомендации по обработке исключений

- Помните, что генерация исключений - процесс, требующий значительных системных ресурсов.
- Не обрабатывайте конкретное исключение или несколько исключений с использованием общего типа *Exception*. Следует классифицировать исключения.
- Не оставляйте блоки *catch* пустыми. Иначе нельзя будет узнать о факте генерации исключения.
- По возможности, не используйте одинаковую обработку разных исключений.

Рекомендации по обработке исключений

- Не создавайте свой класс исключения, эквивалентный по смыслу существующему.
- Не используйте исключения, которые могут ввести в

заблуждение:

```
public void setYear(int year) throws IOException {
if(year <= 0) {
throw new IOException();
}
releaseYear = year;
}
```

Рекомендации по обработке исключений

- Следует проверять на *null*, а не генерировать *NullPointerException*. Если по логике программы необходимо выбросить такое исключение, то больше подойдет *IllegalArgumentException*.
- Избегайте вкладывания блоков *try-catch* друг в друга. Это ведет к ухудшению читаемости кода.
- Не выбрасывайте исключения из блока *finally*.