Обработка ошибок

Croc Java School

Примеры ошибок

- Неправильный индекс при работе с массивом.
- Ошибка доступа к ресурсу: файл не существует, сервер не отвечает, ошибка аутентификации.
- Ошибки VM: переполнение стека, исчерпание памяти, и т.п.
- Пользовательские ошибки.

Исключения

Для сигнализации об ошибках в Java используются специальные объекты — исключения.

```
void setMonth(int month) throws Exception {
    if (month < 0 || month > 12)
        throw new Exception(“Month should be between 1 and 12”);
    this.month = month;
}
```

Генерация исключений

Ключевое слово `throws` сигнализирует, что:

- обработка текущего метода должна быть остановлена
- вызывающий метод должен обработать сгенерированное исключение

Так как работа метода прерывается, возвращаемое значение отсутствует.

```
int alwaysThrow() throws Exception {
    throw new Exception();
    return 7; // error: Unreachable statement
}
```

Обработка исключений

Для обработки исключений используется конструкция `try-catch`.

```
void alwaysIgnore() {
    int result = 0;
    try {
        result = alwaysThrow();
    } catch (Exception e) {
        // ignore error
    }
    // result is always 0 here
}
```

Обработка исключений

```
try {
    // statements that possibly throw an exception
} catch (exception_type identifier) {
    // statements invoked when exception_type is thrown
    // in a try block
}
```

Блок `try` прерывается при генерации первого исключения.

Блок `catch` выполняется, если в блоке `try` было сгенениновано исключение указанного типа.

Фигурные скобки обязательны.

Обработка разных типов исключений

```
try {
    authenticate(user);
} catch (AuthenticationException e) {
    // invalid user credentials
} catch (IOException e) {
    // report service unavailable
}
```

Может выполниться только один из `catch`-блоков.

Порядок следования важен, т.к. типы исключений проверяются именно в этом порядке.

Обработка разных типов исключений

```
try {
    authenticate(user);
} catch (AuthenticationException | IOException e) {
    // handle both types in a same way
}
```

finally

```
try {
    // protected code
} catch (Exception e) {
    // exception handler
} finally {
    // finally block:
    // always executes
}
```

finally

Если в блоке `try` генерируется исключение, `finally`-блок выполняется после блока `catch`.

Если исключение не генерируется — после блока `try`.

Правильный порядок определения: `try-catch-finally`.

Блок `catch` может отсутствовать при наличии блока `finally`.

finally and System.exit

`System.exit` прерывает выполнение Java-процесса.

Блок `finally` не выполняется после вызова `System.exit`.

```
try {
     System.exit(0);
}  finally {
     // not executed
}
```

Работа с ресурсами

```
public class Texture {

    private int handle;

    public Texture(String path) throws Exception {
        handle = allocateTexture(path);
    }

    public void close() throws Exception {
        freeTexture(handle);
    }
}
```

Работа с ресурсами

Корректный сценарий работы с ресурсами: освобождение как в случае успешного “пути”, так и в случае ошибки

```
Texture texture = null;
try {
    texture = new Texture(“tree.png”);
    // use texture
} finally {
    texture.close(); // free texture data
}
```

java.lang.AutoCloseable

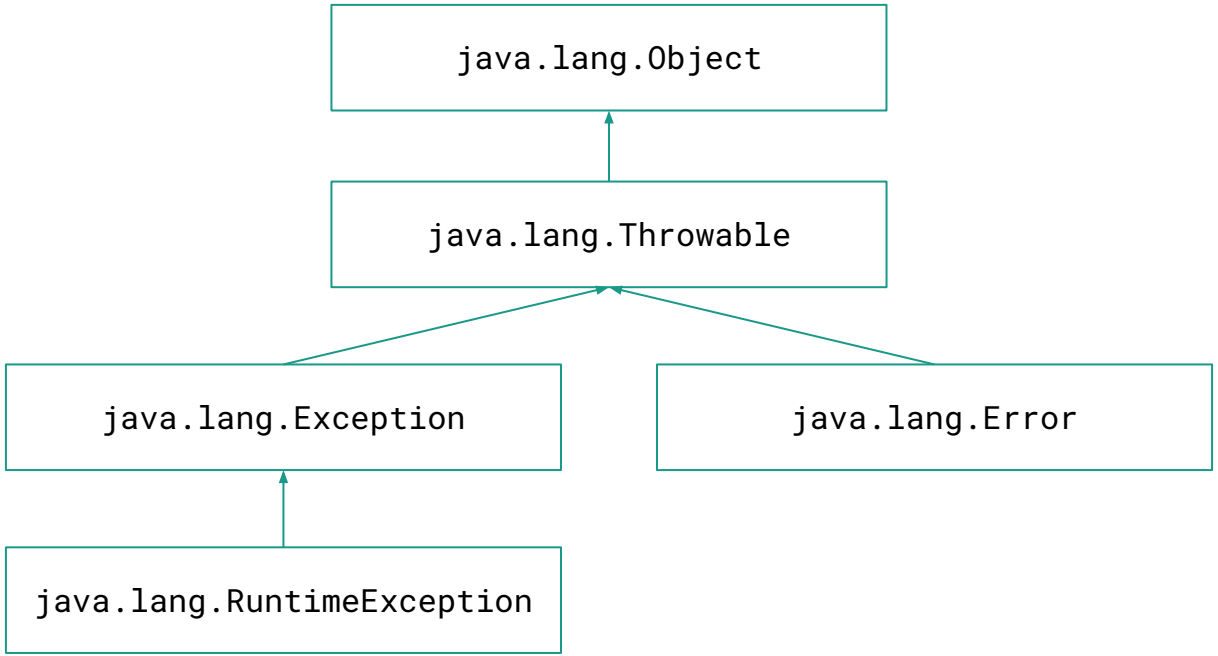
```
public class Texture implements AutoCloseable {

    @Override
    public void close() throws Exception {
        freeTexture(handle);
    }
}
```

try-with-resources

```
try (Texture texture = new Texture(“tree.png”)) {
    // use texture
} finally {
    if (texture != null)
        texture.close();
}
```

try-with-resources применим ко всем типам, реализующим интерфейс `AutoCloseable`.

Типы исключений



Типы исключений

`Error` — ошибка виртуальной машины, приложение не должно предпринимать попыток обработать исключения типа `Error`.

Обрабатываемые (checked) исключения (`Exception`, но не `RuntimeException`) — подчиняются правилу “handle or declare”.

Необрабатываемые (unchecked) исключения.

Правило “handle or declare”

Метод обязан либо обработать исключение (handle) либо передать на вышестоящий уровень (declare).

handle

```
public boolean deleteCache() {
    Path path = Paths.get(“cache.txt”);
    try {
        Files.delete(path);
    } catch (IOException e) {
        return false;
    }
    return true;
}
```

declare

```
public void deleteCache() throws IOException {
    Path path = Paths.get(“cache.txt”);
    Files.delete(path);
}
```

Типы исключений

Что нужно помнить о типах исключений

	Ok to catch	*Must handle or declare*
`Error`	Нет	Нет
`Unchecked`	Да	Нет
`Checked`	Да	Да

Stack trace

```
1 public class Main {
2
3    public static void main(String[] args) {
4        fail();
5    }
6
7    public static void fail() {
8        int[] array = new int[3];
9        array[4] = 1;
10   }
11 }
```

Stack trace

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
    at Main.fail(Main.java:9)
    at Main.main(Main.java:4)
```

Антипаттерн обработки исключений

Пустой `catch`-блок — не решение проблемы, а уход от нее.

```
public void doSomeMagic() {
    try {
        // some complex logic
    } catch (Exception e) {
    }
}
```

catch-and-rethrow

Обрабатываемое исключение может быть переупаковано в необрабатываемое.

```
public void deleteCache() {
  Path path = Paths.get("cache.txt");
  try {
     Files.delete(path);
  } catch (IOException e) {
     throw new RuntimeException(e);
  }
}
```

Кастомные исключения

```
public class PasswordTooShortException extends Exception {

  private final int length;
  private final int minLength;

  public PasswordTooShortException(int length, int minLength) {
     this.length = length;
     this.minLength = minLength;
  }

  @Override
  public String getMessage() {
     return "Password length should be at least " + minLength;
  }

  // getters
}
```

Кастомные исключения (использование)

```
public void changePassword(String password)
     throws PasswordTooShortException {

if (password.length() < MIN_PASSWORD_LENGTH) {
     throw new PasswordTooShortException(
          password.length(), MIN_PASSWORD_LENGTH);
  }
}
```

Сигнатуры конструкторов, к которым все привыкли

Стоит определить хотя бы один конструктор, принимающий cause

```
public class Exception extends Throwable {

  public Exception()

  public Exception(String message)

  public Exception(String message, Throwable cause)

  public Exception(Throwable cause)

  ...
}
```