

**Жизненный цикл ПО.
Профессии в IT.
Виды приложений**

Жизненный цикл ПО

Цикл разработки ПО

1. Возникновение идеи или проблемы у заказчика.

Заказчик осознал, что у него есть какая-то потребность или проблема, и что для её решения нужно разработать ПО

2. Сбор и анализ требований.

Формирование ТЗ (технического задания).

ТЗ – это документ, в котором описано, что именно будет разрабатываться, и в котором описаны требования.

Заказчик может написать ТЗ сам, либо совместно с исполнителем (компанией по разработке).

Так или иначе ТЗ составляется и согласовывается

Цикл разработки ПО

3. Проектирование и планирование.

Проектируется архитектура проекта, какие технологии использовать и т.д.

Менеджер проекта (человек, ответственный за выполнение проекта) оценивает стоимость, трудозатраты и сроки выполнения проекта

Цикл разработки ПО

4. Разработка.

- Дизайн (оформление)
- Кодирование (написание кода)
- Тестирование (проверка на соответствие требованиям)
- Документирование

5. Внедрение.

Передача заказчику, установка ПО пользователям

6. Сопровождение.

Консультирование заказчика по вопросам.

Исправление ошибок (за счет исполнителя).

Доработка функционала (за отдельную плату)

7. Вывод из эксплуатации.

Методологии разработки

- Мы рассмотрели основные этапы жизненного цикла ПО
- Но как именно поставить эти процессы в проекте?
- Для этого есть множество различных **методологий разработки**
- Старейшей, и самой простой для понимания, является **каскадная модель** (или **водопад**)

Каскадная модель разработки

- Все этапы выполняются по очереди



Каскадная модель разработки

- **Минусы:**
 - Конечный продукт получается только в конце
 - При изменении требований приходится откатываться назад
- **Плюсы:**
 - Простота и предсказуемость сроков и бюджета
- **Итого:**
 - Подходит в проектах, в которых требования понятны и зафиксированы
- Большая часть проектов под эти требования не попадает, поэтому там эта методология неэффективна. В большинстве проектов используют **гибкие методологии**

Гибкие методологии (Agile)

- Чем они отличаются от каскадной?
 - Мы смиряемся с тем, что **требования постоянно меняются**. И мы готовы к этому
 - Частая коммуникация с заказчиком
 - Продукт получается не в конце, а поставляется поэтапно. И уже может приносить пользу заказчику



Scrum

- Все задачи по проекту вносят в так называемый **backlog** – список всех задач, которые есть по проекту
- Разработка делается небольшими итерациями (**спринтами**) длительностью в несколько недель
- Для каждого спринта выбирается перечень задач из backlog'а, которые надо сделать в рамках этого спринта
- Перечень выбирается совместно с заказчиком, расставляются приоритеты
- Каждый день проводятся небольшие митинги – кто чем занят, у кого какие проблемы есть и т.д.
- После завершения спринта проводится **ретроспектива**

Литература

- Про Scrum:
- <https://goo.gl/MgWjbN>
- Про Scrum и Kanban:
- <http://rulesplay.ru/materialy/statii/skram-ili-kanban/>
- Раздел статей:
- <https://www.atlassian.com/ru/agile>
- Статей очень много, ищите и другие

Профессии в IT

Профессии в IT

- Программист (разработчик)
- Тестировщик (контроль качества, QA (Quality Assurance))
- Системный аналитик
- Менеджер проекта
- Team lead
- Технический писатель
- Дизайнер
- HR (human resources)
- Системный администратор
- И др.

Программист

- Разрабатывает программу в соответствии с требованиями проекта

Тестировщик

- Контролирует качество ПО
- При тестировании проверяется, что программа соответствует **требованиям** заказчика
- Требования описаны в специальных документах – **техническое задание** и **функциональная спецификация**
- К сожалению, часто этих документов нет, а требования передаются разработчикам от заказчика на словах, что сильно ухудшает жизнь
- Без этих документов сложно проверить, что программа соответствует требованиям, потому что они нигде не зафиксированы
- И сложно вводить в курс дела других людей

Тестировщик

- **Баг (ошибка)** – это расхождение между желаемым результатом и реальным результатом
- Если программа не соответствует требованиям заказчика, то это баг
- Кроме того, багом являются ошибки программы. Например, что программа падает при некоторых случаях
- Кроме того, тестировщики обращают внимание на **usability – удобство использования программы**
- Тестировщики обращают внимание, если некоторые части программы работают медленно, либо не дают пользователю отклик, либо можно сделать более удобный пользовательский интерфейс

Тестировщик

- Тестирование бывает **ручным** и **автоматизированным**
- **Ручное тестирование** – тестировщик проходит кейсы вручную
- **Автоматизированное тестирование** – тестировщик использует инструменты для автоматизации тестирования или пишет специальные программы-тесты, которые запускаются автоматически
- Автоматизированное тестирование - более сложная область, нужно знать и тестирование, и программирование. Такие тестировщики очень ценятся

Системный аналитик

- **Системный аналитик** занимается:
 - Сбором, анализом и систематизацией требований заказчика
 - Составлением ТЗ и других документов
 - Изучением предметной области проекта, бизнеса заказчика, потребностей пользователя
- Системный аналитик должен перевести требования заказчика в язык, понятный разработчикам
- И наоборот – донести до заказчика информацию от разработчиков на понятном заказчику языке

Дизайнер

- Придумывает как будет выглядеть интерфейс программы – цвета, оформление и др.
- Создает изображения интерфейса в нескольких вариантах:
 1. Удобный для заказчика – просто картинка
 2. Удобный для разработчика – например, сделана нарезка отдельных картинок, подписаны цвета, размеры и отступы
- Многие дизайнеры знают какие элементы управления лучше использовать, т.е. разбираются в **usability**

HR (Human Resources)

- Занимаются поиском новых сотрудников, размещают вакансии, договариваются с техническими специалистами насчет собеседований
- В некоторых компаниях есть отдельный этап собеседования – только с представителем HR, чтобы сразу отсеять людей, которые не подходят по личностным качествам
- Если вы подаете резюме на вакансию, то потом с вами связывается именно HR
- Также HR отвечает за работу с уже нанятыми сотрудниками, и за имидж компании как работодателя

Системный администратор

- Обеспечивает работоспособность компьютеров, сети и программных продуктов
- Умеет администрировать многие продукты, например операционные системы, почтовые и веб-серверы, базы данных и др.
- Также следит, чтобы сотрудники не ставили нелицензионное ПО
- Следит за безопасностью инфраструктуры и ПО
- Может участвовать во внедрении продуктов

Технический писатель

- Занимается разработкой и поддержкой документации
- Хорошо владеет русским и, желательно английским языками. Имеет опыт работы в сфере IT, а желательно и разработки ПО
- Достаточно редкая профессия, но очень нужная, потому что без них документы никто не ведет
- Это ведет к большим проблемам при передаче проектов и опыта между сотрудниками и анализе требований к программе

Менеджер проекта

- **Менеджер проекта** – это человек, ответственный за успешное завершение проекта
- Обеспечивает, что проект будет завершен в срок, уложится в запланированный бюджет, и будет надлежащего качества
- Менеджер должен быть лидером, уметь налаживать работу команды, обеспечивать команду всем необходимым
- Кроме того, менеджер занимается планированием сроков и бюджета, общением с заказчиком
- Менеджер должен уметь налаживать коммуникации между всеми членами команды

Team lead и менеджер проекта

- **Team lead** – руководитель команды разработки. Является программистом, участвует в написании кода, частично или полностью выполняет функции менеджера проектов
- **Team lead** лучше понимает устройство программы, но не может вести много проектов одновременно
- Часто менеджерами проектов являются и не разработчики. Тогда они не участвуют в написании кода, и больше занимаются общением с заказчиком и др.
- Такие менеджеры могут вести много проектов одновременно, а по техническим вопросам обращаются к команде разработчиков

Варианты карьеры тестировщика

- **Повышение экспертизы:**
 - Junior QA -> Middle QA -> Senior QA
- **Переход в управление QA:**
 - QA -> QA Team Lead (руководитель команды тестировщиков) -> Руководитель QA в компании
- **Переход в автоматизацию:**
 - QA -> Тестировщик-автоматизатор
- **Переход в аналитику:**
 - QA -> Системный аналитик

Переход тестировщика в разработку

- Переход тестировщика в разработку маловероятен
- Почему так?
- Дело в том, что предыдущие варианты являются гармоничным развитием на позиции тестировщика
- Например, вы тестировщик, и изучили инструменты автотестирования и получили базовые навыки программирования – вы становитесь автоматизатором
- Или вы развиваете soft-skill'ы, и переходите в управление
- Или пишете ТЗ, и переходите в аналитику
- Часто эти навыки вы приобретаете сами собой во время работы
- При этом ваша зарплата, скорее всего, вырастет

Переход тестировщика в разработку

- Чтобы перейти в разработку даже на Junior'а нужно достаточно глубокое знание программирования
- Эти навыки нельзя получить сами собой во время работы тестировщиком – нужно будет учиться программированию отдельно от работы
- Естественно, опыт работы в тестировании будет полезен, но он не является определяющим
- При этом при переходе, скорее всего, будет падение зарплаты, т.к. вы станете Junior разработчиком

Переход тестировщика в разработку

- Компании очень не любят когда кандидаты на тестировщика говорят, что потом планируют уйти в разработку
- Т.к. тогда возникает вопрос, а почему тогда не идете в разработку сразу
- Компания хочет найти того, кому будет интересно тестирование, поэтому такие кандидаты сразу отсеиваются
- В общем, это никогда нельзя говорить на собеседовании тестировщика
- При этом если вы уже работаете в компании тестировщиком достаточно долгий срок, выучили программирование, то тогда к переходу отнесутся нормально

Варианты карьеры разработчика

- **Повышение экспертизы:**
 - Junior -> Middle -> Senior
- **Переход в управление разработкой:**
 - Developer -> Team Lead (руководитель команды разработки) -> Технический директор
- **Переход в автоматизацию тестирования:**
 - Developer -> Тестировщик-автоматизатор
- **Переход в архитекторы:**
 - Developer -> Architector

Виды приложений

Виды приложений

- Приложения можно разделить на следующие виды:

- **Десктопные (оконные)**
- Консольные
- **Веб-приложения (сайты)**
- **Мобильные приложения**
 - **Android**
 - **iOS**

Самые распространенные приложения – это сайты и мобильные приложения на Android и iOS

Чуть менее распространены десктопные приложения

Остальное менее приоритетно

Десктопные приложения

- **Десктопные приложения** – это привычные оконные приложения
- Имеют графический интерфейс
- Например: Microsoft Word, Skype

Консольные приложения

- **Консольные приложения** – имеют текстовый интерфейс
- Сюда относятся:
 - Команды и полезные программы операционной системы: `dir`, `cd`, `rm` и т.д. **Показать пример**
 - Различные утилиты для разработчиков и тестировщиков:
 - Средства автоматизации разработки и тестирования
 - Вспомогательные программы, которые требуется запустить всего 1 раз, и поэтому не хочется тратить время на создание полноценного графического интерфейса

Веб-приложения

- **Веб-приложения или сайты** – это распределенные приложения (располагаются на нескольких компьютерах), имеющие **клиент-серверную архитектуру**
- Понятие клиент-серверной архитектуры рассмотрим позже

Мобильные приложения

- **Мобильные приложения** – приложения для мобильных устройств
- Распространенные мобильные ОС:
 - Android
 - iOS
- Для каждой мобильной ОС обычно пишут отдельное приложение
- Каждая ОС отличается по своей концепции и гайдлайнам (стандартам оформления пользовательских интерфейсов), которые нужно знать

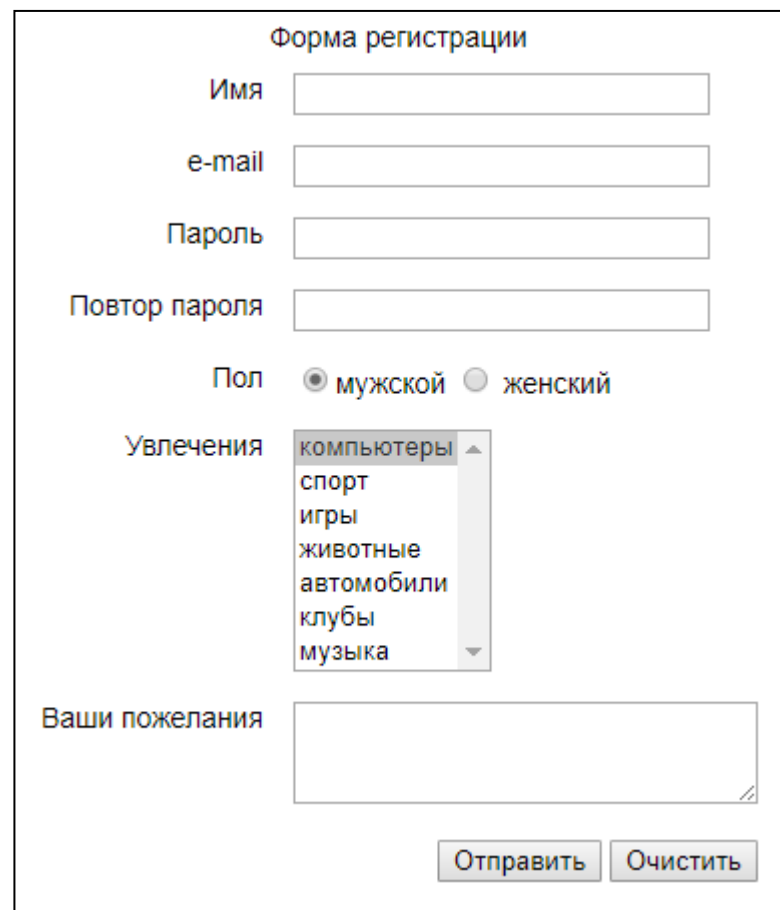
Пользовательский опыт

- Для тестировщика важно иметь большой пользовательский опыт в использовании приложений тех типов, с которыми он работает
- Вы должны знать какие есть элементы интерфейса (кнопки, однострочные поля ввода, чекбоксы и т.д.), как они называются, в чем их роль, как они себя ведут
- Должны знать хорошие практики по построению интерфейсов и плохие практики
- Должны знать хорошие и плохие практики по поведению программы, например, при обрыве соединения или передаче плохих данных
- Все это относится к **usability** (удобство использования для пользователя)

Элементы управления в интерфейсах

Формы

- **Форма** – это часть пользовательского интерфейса, внутри которой есть элементы управления, куда пользователь может ввести данные
- После ввода данных пользователь может **отправить форму**, и тогда данные отправятся на обработку



Форма регистрации

Имя

e-mail

Пароль

Повтор пароля

Пол ☒ мужской ☐ женский

Увлечения

компьютеры

спорт

игры

животные

автомобили

клубы

музыка

Ваши пожелания

Элементы управления в формах

- Кнопки
- Поля ввода:
 - Однострочные
 - Многострочные
- Выпадающие списки:
 - С единичным выбором
 - С множественным выбором
- Чекбоксы (галочки/флажки)
- Радио-баттоны (радио-кнопки)
- Про HTML формы:
<https://webref.ru/course/html-content/forms>

Поля ввода

- Бывают однострочные и многострочные
- Однострочное поле ввода (textbox) позволяет ввести только одну строку текста

Имя

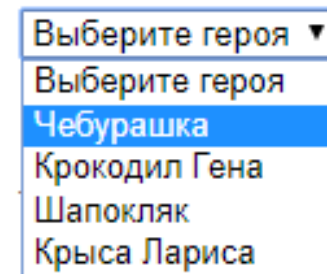
- Многострочное поле ввода (textarea) позволяет вводить много строк текста

Ваши пожелания

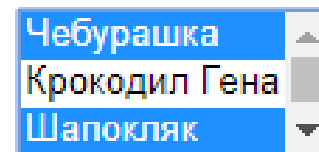
Выпадающие списки

- **Выпадающие списки**, еще иногда называют **dropdown**
- Бывают с единичным выбором (можно выбрать ровно 1 элемент), бывают с множественным выбором (можно выбрать несколько элементов, обычно через Ctrl)
- С единичным выбором

Выберите героя ▼



- С множественным выбором



Чекбоксы

- **Чекбокс (галочка, флажок, крыжик)** – позволяет выбрать один из двух вариантов – Да или Нет
- Здесь пример формы с пятью чекбоксами. Все чекбоксы независимы друг от друга – можно выбрать любую их комбинацию

С какими операционными системами вы знакомы?

- ☒ Windows 95/98
- ☐ Windows 2000
- ☐ System X
- ☐ Linux
- ☐ X3-DOS

Отправить

Радио-баттоны

- **Радио-баттоны (радио-кнопки)** – позволяют выбрать ровно один из нескольких вариантов
- При выборе одной из кнопок, другие сбрасываются
- Радио-баттоны используются если вариантов выбора мало. Иначе используют выпадающий список
- Здесь пример формы с пятью радио-баттонами. Можно выбрать только один из вариантов

Ваша любимая операционная система?

- ☒ Windows 95/98
- ☐ Windows 2000
- ☐ System X
- ☐ Linux
- ☐ X3-DOS

Отправить

Заблокированные элементы

- Элементы управления могут быть заблокированы (**disabled**) или разблокированы (**enabled**)
- Слэнг: заблокировать элемент – бывает, говорят **задизэйблить**, разблокировать – **раздизэйблить**
- Здесь если не делаем рассылку подписчикам (чекбокс не выбран), то нельзя ввести комментарий (textarea задизэйблена)

Подписчики ☐ Комментарий для отправки

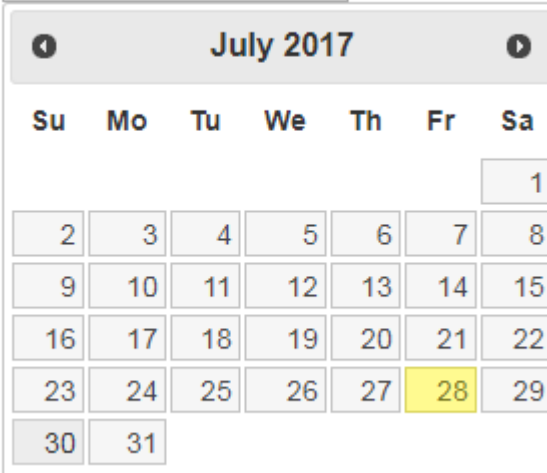
Сложные элементы управления

- **Календарь** (datepicker)
- **Комбобокс** – это выпадающий список, который позволяет ввести свое значение
- **Автокомплит** (autocomplete) – это выпадающий список с поиском
- **Слайдер**
- **Загрузчик файлов** (обычно имеет drag and drop)
- И др.

Календарь (datepicker, дэйтпикер)

- Обычно это текстовое поле, при клике по которому открывается календарь
- <https://jqueryui.com/datepicker/>
- Нужно следить за форматами дат для разных языков
 - Например, в РФ принято писать даты так: 20.01.2017
 - В США так: 01/20/2017

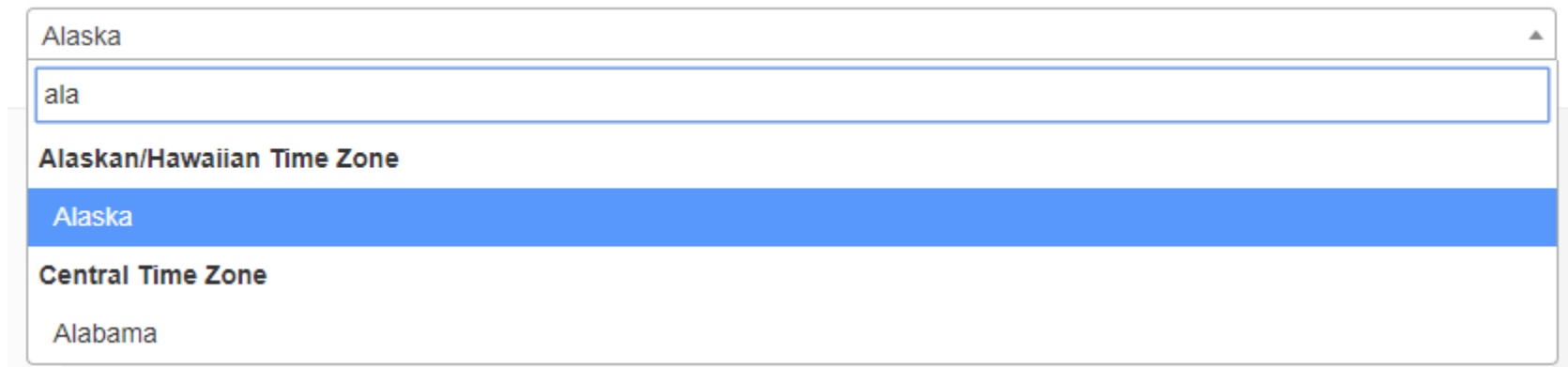
Date:



July 2017						
Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Автокомплит

- Это выпадающий список с поиском
- <https://select2.github.io/examples.html>



The image shows a Select2 dropdown menu. The search input at the top contains the text "Alaska". Below the input, a list of search results is displayed. The first result is "Alaskan/Hawaiian Time Zone". The second result, "Alaska", is highlighted with a blue background. Below it is "Central Time Zone", and at the bottom is "Alabama".

Search Results
Alaskan/Hawaiian Time Zone
Alaska
Central Time Zone
Alabama

Другие элементы интерфейса

- Индикатор загрузки, спиннер (лоадер, крутилка)
 - Сообщает пользователю, что сейчас выполняется длительная операция, надо подождать
 - Хороший стиль, чтобы он был при всех длительных операциях
- Диалог
 - <https://jqueryui.com/dialog>
 - **Модальный диалог** – диалог, который блокирует работу с родительским окном, пока пользователь его не закроет
- Вкладки (табы)
 - <https://jqueryui.com/tabs/>

Форматы XML и JSON

Форматы данных XML и JSON

- Данные между клиентом и сервером чаще всего передаются в форматах XML и JSON
- Оба формата текстовые и легко читаются человеком
- Обычно браузер и мобильные приложения используют JSON
- XML используется при клиент-серверном взаимодействии между разными системами
- XML широко используется для конфигурирования программ, хранения настроек

Форматы данных XML и JSON

- ```
<?xml version="1.0" encoding="UTF-8"?>
<note>
 <to>Tove</to>
 <from>Jani</from>
 <heading>Reminder</heading>
 <body>Don't forget me this weekend!</body>
</note>
```

XML

- ```
{
  "note":{
    "to":"Tove",
    "from":"Jani",
    "heading":"Reminder",
    "body":"Don't forget me this weekend!"
  }
}
```

JSON

XML

- Что почитать:
- <https://msiter.ru/tutorials/uchebnik-xml-dlya-nachinayushchih>
- <https://ru.wikipedia.org/wiki/XML>
- ```
<?xml version="1.0" encoding="UTF-8"?>
<note>
 <to>Tove</to>
 <from>Jani</from>
 <heading>Reminder</heading>
 <body>Don't forget me this weekend!</body>
</note>
```

# Объявление XML

- XML документ начинается с **объявления XML**
- Это первая строка в примере ниже
- Там указывается версия формата XML и кодировка
- ```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

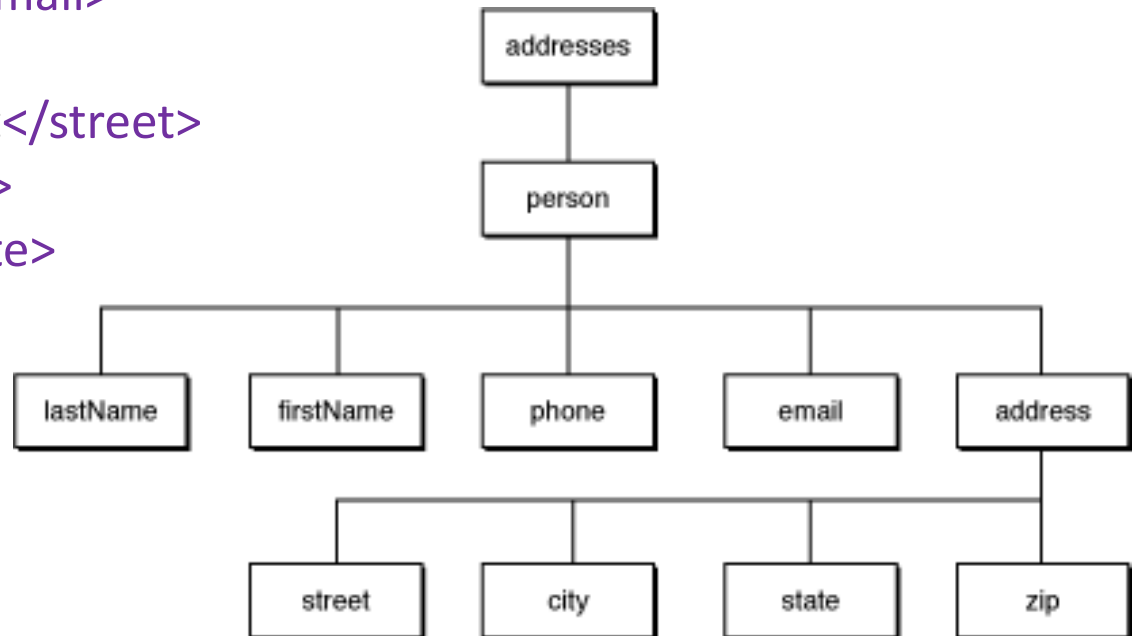
Корневой элемент, дерево

- XML документ (как кстати и JSON) представляет собой **дерево**
- У этого дерева есть **корень** – элемент верхнего уровня, который никуда не вложен
- В данном примере это элемент **note**
- В note вложены **дочерние элементы** to, from, heading, body
- ```
<?xml version="1.0" encoding="UTF-8"?>
<note>
 <to>Tove</to>
 <from>Jani</from>
 <heading>Reminder</heading>
 <body>Don't forget me this weekend!</body>
</note>
```

У них тоже могут быть вложенные элементы и т.д.

# Корневой элемент, дерево

```
<?xml version="1.0" encoding="UTF-8"?>
<addresses>
 <person idnum="0123">
 <lastName>Doe</lastName>
 <firstName>John</firstName>
 <phone location="mobile">(201) 345-6789</phone>
 <email>jdoe@foo.com</email>
 <address>
 <street>100 Main Street</street>
 <city>Somewhere</city>
 <state>New Jersey</state>
 <zip>07670</zip>
 </address>
 </person>
</addresses>
```



# Теги

- Элементы описывают при помощи **тегов**
- Это метки вида `<note>` или `</note>`
- `<note>` - это открывающий тег, с него начинается элемент
- `</note>` - это закрывающий тег, на нем заканчивается элемент
- Все что находится внутри является содержимым элемента `note`
- `<?xml version="1.0" encoding="UTF-8"?>`  
`<note>`  
    `<to>Tove</to>`  
    `<from>Jani</from>`  
    `<heading>Reminder</heading>`  
    `<body>Don't forget me this weekend!</body>`  
`</note>`

Такие теги называют парными



# Одиночные теги

- Могут быть **одиночные теги**, которые не требуют парного закрывающего тега
- В данном примере это тег `<important />`
- Обратите внимание на слэш в конце, он обозначает, что тег одиночный. Еще его называют **самозакрывающим тегом**
- ```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
  <important />
</note>
```

Атрибуты

- У тегов могут быть **атрибуты**, это дополнительные параметры тегов
- Их можно писать только в открывающих и самозакрывающихся тегах
- У атрибута есть имя и есть значение, которое указывают после =
- Например, тут у note есть атрибут number со значением 1
- ```
<?xml version="1.0" encoding="UTF-8"?>
<note number="1">
 <to>Tove</to>
 <from>Jani</from>
 <heading>Reminder</heading>
 <body>Don't forget me this weekend!</body>
</note>
```

# Атрибуты

- У элемента может быть много атрибутов, порядок не важен
- Значение заключают в двойные кавычки, они обязательны
- Пробелы вокруг = не ставят
- ```
<?xml version="1.0" encoding="UTF-8"?>
<note number="1" important="true">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

Специальные последовательности

- Если в документе встретятся эти символы, то разметка станет некорректной: >, <, &
- Чтобы все-таки их можно было указывать в документе, их нужно заменять на специальные последовательности:

Символ	Замена
<	<
>	>
&	&

- ```
<?xml version="1.0" encoding="UTF-8"?>
<note>
 <body>Don't forget <me> & her!</body>
</note>
```
- По смыслу будет: Don't forget <me> & her!

# Специальные последовательности

- Аналогично в атрибутах запрещены кавычки: ' и "
- Их нужно заменять так:

Символ	Замена
'	&apos;
"	&quot;

- ```
<?xml version="1.0" encoding="UTF-8"?>
<note tag="&quot;me&quot;">
  <body>Don't forget me!</body>
</note>
```

CDATA

- Есть и другой способ как сделать чтобы <, >, & обрабатывались как текст
- Это секция **CDATA**
- Она пишется так: `<![CDATA[содержимое]]>`
- ```
<?xml version="1.0" encoding="UTF-8"?>
<note>
 <body><![CDATA[<sender>Иван</sender>]]></body>
</note>
```
- Все что идет внутри CDATA, будет обработано так, будто это просто текст, а не теги. Т.е. будто там &lt; и др.
- <https://ru.wikipedia.org/wiki/CDATA>

# Важные моменты для понимания XML

- Названия тегов и атрибутов в XML не являются predetermined в отличие, например, от HTML
- Тот, кто придумывает формат конкретного XML документа, сам придумывает все названия тегов и саму структуру вложенности элементов
- XML сам по себе ничего не делает
- Это просто способ записи данных в виде текста в структурированном виде
- Но программы могут читать этот XML и вытаскивать из него данные, и работать с этими данными

# Типы данных

- В программировании у всех данных есть **тип данных**
- Примеры:
  - **Целое число:** 310
  - **Вещественное число (десятичная дробь):** 23.34354
  - **Строка:** Текст
  - **Дата:** 2010-03-10
  - **Время:** 12:03:00
  - **Дата и время:** 2010-03-10 12:03:00
  - **Логический тип (boolean):** **true** или **false** — истина или ложь



# Типы данных

- Типы данных должны соблюдаться, иначе программа не сможет распознать формат
- Частая ошибка при передаче данных – когда данные находятся не в том формате
- Например, требуется число, а передали строку
- Или требуется целое число, а передали вещественное
- Или ошиблись в написании значений `true`, `false` и т.д.

# Практика

- Давайте разберем какие сущности описаны в документе, какие у них есть характеристики, какие у них типы данных (число, строка и т.д.)

- `<?xml version="1.0" encoding="utf-8" ?>`

`<users>`

`<user name="Bill Gates">`

`<company>Microsoft</company>`

`<age>48</age>`

`</user>`

`<user name="Larry Page">`

`<company>Google</company>`

`<age>48</age>`

`</user>`

`</users>`

Заметим, что у элементов `user` одинаковая структура.

Тут просто 2 элемента одного типа данных `user`

**JSON**

# JSON

- Формат JSON в отличие от XML не требует в начале строку с объявлением, данные описываются сразу же
- ```
{  
  "name": "Ivan",  
  "age": 30,  
  "cat": {  
    "name": "Murka"  
  }  
}
```
- Сам синтаксис более лаконичный – не требуются закрывающие теги
- За счет этого объем данных сокращается, что удобно для передачи по сети

JSON – что почитать

- <https://ru.wikipedia.org/wiki/JSON>
- <http://www.wisdomweb.ru/AJAX/json.php>

JSON

- Формат JSON поддерживает следующие типы данных:
 - **Числа (целые и вещественные):** 3 4.4
 - **Логический тип (boolean):** true/false (истина/ложь)
 - **Строки** – заключаются в двойные кавычки
 - “Строка 1”
 - **Объекты** – сущности, у которых могут быть **поля** (характеристики этих сущностей)
 - {“name”: “Ivan”, “age”: 30, “cat”: {“name”: “Murka”}}
 - **Массивы** – списки значений через запятую
 - [1, 2, 5, 4]
 - **Значение null** – отсутствие данных

JSON объекты

- **Объекты** – сущности, у которых могут быть **поля** (характеристики этих сущностей)
- ```
{
 "name": "Ivan",
 "age": 30,
 "cat": {
 "name": "Murka"
 }
}
```
- Название поля и его значение отделяют двоеточием :
- Поля могут быть любых типов
- Имена полей должны быть заключены в двойные кавычки
- Порядок полей не важен, друг от друга их отделяют запятыми

# JSON объекты

- В одном объекте не может быть полей с одинаковым именем



# Массивы

- **Массив** – это список элементов
- JSON позволяет добавлять в список элементы разных типов, но так обычно не делают, т.к. важна универсальность и простота структуры
- Поэтому считайте, что массив должен хранить элементы одного и того же типа
- Например, массив целых чисел: [1, 3, 4, 10]
- Или массив объектов: [{ “name”: “Petr” }, { “name”: “Irina” }]

# Экранирование

- В значениях нельзя просто так использовать следующие символы: \, /, “
- Чтобы их можно было использовать, их надо **экранировать** – написать перед ними обратный слэш \
- Этот экранирующий обратный слэш будет игнорироваться
- { “url”: “http:\\\\google.ru” } // <http://google.ru>
- { “text”: “\\”Кавычки\\”” } // “Кавычки”
- { “path”: “C:\\\\Windows\\system” } // C:\\Windows\\system

# Пример

- JSON:
- [https://api.vk.com/method/groups.getById?group\\_id=122100659&v=5.74](https://api.vk.com/method/groups.getById?group_id=122100659&v=5.74)
- Отформатируем его при помощи онлайн форматера:
- <https://jsonformatter.curiousconcept.com/>
- Давайте разберем какие сущности описаны, какие у них есть характеристики, какие у них типы данных (число, строка и т.д.)

# Соглашения именования

- В программировании важно соблюдать соглашение именования – в каком стиле даются имена
- Есть несколько распространенных соглашений:
  - **Верблюжья нотация (camel case)** – слова пишутся без пробелов, каждое последующее слово с заглавной буквы, по аналогии с горбами верблюда
    - **Примеры:** countryCode, numberOfPeople
  - **Нотация с нижним подчеркиванием** - слова пишутся через \_, маленькими буквами
    - **Примеры:** country\_code, number\_of\_people
- Возможны и некоторые другие варианты – например, верблюжья нотация с заглавной буквы

# Соглашения именования

- **Верблюжья нотация (camel case)** применяется в JSON и многих языках программирования. Можно применять ее и в XML
  - **Примеры:** countryCode, numberOfPeople
- **Нотация с нижним подчеркиванием** – обычно применяют только в XML
  - **Примеры:** country\_code, number\_of\_people
- Не важно какой стиль вы выберете для XML, главное - единообразие

# Домашнее задание

# Домашняя задача «Элементы форм»

- Откройте сайт <http://auto.drom.ru/>
- Зайдите в расширенный поиск, откроется форма поиска
- Сделайте скриншот формы и подпишите каждый элемент как он называется. Например, чекбокс, радио-баттон и т.д.
- Смысл задания – надо знать названия всех элементов форм

# Домашняя задача «Анализ формы»

- <http://auto.drom.ru/>
- Проанализируйте форму из предыдущего задания и выпишите логику её работы. Какие есть зависимости между элементами?
- Возможно, что-то сбрасывается при выборе чего-то другого. Возможно, что-то появляется при выборе чего-то, или дизэйблится
- Надо найти и выписать все такие правила.  
Учтите и саму кнопку-ссылку «Расширенный поиск»
  - Смысл задания - развиваем умение понимать логику работы программы, не имея на руках ТЗ



# Домашняя задача «Прочитать XML»

- Опишите что за сущности описаны в данном XML документе
- [https://msdn.microsoft.com/ru-ru/library/ms762271\(v=vs.85\).aspx](https://msdn.microsoft.com/ru-ru/library/ms762271(v=vs.85).aspx)
- Какие у них есть характеристики? Напишите смысл каждой характеристики на русском
- Какой тип данных у каждой из этих характеристик? (например, число, строка, логический тип – true/false и т.д.)

# Домашняя задача «Прочитать JSON»

- **Часть 1.** Опишите что за сущности описаны в JSON документе **countries.json**, который выложен в группе
- Какие у них есть характеристики? Напишите смысл каждой характеристики на русском (если какие-то не понимаете, то пропустите)
- Какой тип данных у каждой из этих характеристик? (например, число, строка, логический тип – true/false и т.д.)
- Воспользуйтесь JSON-форматером чтобы более красиво отформатировать JSON:  
<https://jsonformatter.curiousconcept.com/>

# Домашняя задача «Прочитать JSON»

- **Часть 2.** Ответьте на вопросы по данным из файла **countries.json**, который выложен в документах группы
- Какие языки являются государственными в Канаде?
- Какая валюта в Мексике?
- Какая столица в Перу?
- Какие часовые пояса в Бразилии?
- С какими странами граничит Аргентина?

# Домашняя задача «XML и JSON»

- Создать XML и JSON файлы, которые описывают список стран с городами
- В документе должно быть 3-5 стран, для них должны быть следующие данные: название на русском, название на английском, код страны, код валюты и список из 2-3 городов этой страны
- Для каждого города нужно будет указать название на русском, название на английском и численность населения
- В качестве текстового редактора удобно использовать IntelliJ IDEA или Visual Studio
  - <https://www.jetbrains.com/idea/download/>
- Файл с XML должен иметь расширение .xml, файл с JSON - .json