

## **Почему именно Java. История создания области использования. Состав и платформы Java.**

Java — строго типизированный ОО язык программирования общего назначения. Разработан: Sun Microsystems (1991). Разработка ведётся через Java Community Process; язык и технологии по лицензии GPL. Права на торговую марку принадлежат корпорации Oracle.

**Области применения JAVA:**

1. Графический интерфейс: фреймворки: Swing и JavaFX.
2. Enterprise: тяжелые серверные приложения для банков, корпораций, инвестфондов и т.д.
3. Mobile: мобильная разработка (телефоны, планшеты), благодаря Android.
4. Web: лидирует PHP, но и Java: фреймворков Struts и JSP (серверные страницы Java)
5. Big Data: распределенные вычисления в кластерах из тысяч серверов.
6. Smart Devices: программы для умного дома, электроники, холодильников с выходом в интернет

**Состав:**

1. JVM (Java Virtual Machine) отвечает за исполнение Java-программ. Основа для достижения кроссплатформенности, поскольку является прослойкой между операционной системой и Java-программой;
2. JRE (Java Runtime Environment) создает и запускает JVM, т.е. представляет собой пакет инструментов для запуска Java-кода. Может использоваться как отдельный компонент для простого запуска Java-программ. ;
3. JDK (Java Development Kit) позволяет разработчикам создавать программы, которые могут выполняться и запускаться посредством JVM и JRE (требует, потому что запуск программ – это часть разработки)

## **Преимущества и основные характеристики Java. В чем заключается кроссплатформенность.**

**Преимущества:**

1. Кроссплатформенность т.к. полная независимость байт-кода
2. Безопасность - исполнение программы полностью контролируется виртуальной машиной.
3. Один из самых доступных языков для обучения и ОО язык
4. В Java реализованы механизмы распределенных приложений
5. Многопоточность – т.е. процесс, может состоять из потоков выполняющихся параллельно без предписанного порядка во времени
6. Сборщик мусора - Автоматически высвобождает неиспользуемую память

В Java доступны следующие платформы:

Java Enterprise Edition (Java EE) - для создания коммерческих приложений для крупных и средних предприятий,

Java Standard Edition (Java SE) для создания и исполнения приложений для рабочих станций (или использования в масштабах малого предприятия)

Java Mobile Edition (Java ME) – для разработки приложений для устройств, ограниченных в ресурсах (сотовых телефонов, карманных персональных компьютеров и т.п.).

# Структура программы на языке Java. Типы данных. Что такое пакет.

## Основные пакеты и стандартные классы в Java.

Программа на языке Java состоит из:

1. Директивы import и package
2. Объявления класса class. При этом исходный код программы хранится в текстовом файле с расширением \*.java, а имя файла соответствует названию класса т.е. для класса MyClass файл будет называться MyClass.java. Класс, с которого начинается выполнение приложения java принято называть главным классом (main class)
3. Объявления и инициализации атрибутов класса. При этом в одном файле может существовать только один публичный (обозначенный модификатором public) класс и его название должно соответствовать имени файла. Код класса ограничен парой фигурных скобок {}.
4. Объявления методов. Работа приложения Java начинается с выполнения главного метода main() одного из классов. Метод принимает на вход массив параметров командной строки
5. Комментарии.

Пакет (package) – пространство имен в Java

Пакет объединяет типы (классы, интерфейсы, перечисления), относящиеся к одной предметной области или одной задаче.

Предопределенные пакеты в java-это те, которые разработаны SunMicrosystem. Они также называются встроенными пакетами в java. Эти пакеты состоят из большого количества предопределенных классов, интерфейсов и методов, которые используются программистом для выполнения любой задачи в его программах.

Стандартные классы Java:

- |                       |                       |
|-----------------------|-----------------------|
| ● java.lang.String    | ● java.util.Random    |
| ● java.lang.Math      | ● java.io.PrintWriter |
| ● java.lang.Integer   | ● java.io.File        |
| ● java.lang.Thread    | ● java.awt.Frame      |
| ● java.util.ArrayList | ● java.awt.Button     |

Классы и пакеты в файловой системе

- Один тип (класс/интерфейс) может принадлежать только одному пакету
- Один файл исходного кода может содержать только один публичный (public) тип, доступный за пределами пакета
- Рекомендуется создавать отдельный файл исходного кода под отдельный тип
- Файл исходного кода должен называться по имени публичного типа
- Переменная окружения CLASSPATH определяет пути, по которым приложения Java будут искать пользовательские классы
- Имя пакета ассоциируется с иерархией папок в файловой системе
- Рекомендуется использовать в названии пакета только символы нижнего регистра
- Стандартные пакеты java начинаются со слов java. и javax.
- Внешние компании используют для именования пакетов свои доменные имена (например, com.ibm.) - package ru.bstu.it32.kurbatova.lab1;

\*Архив Java – JAR – архив в формате ZIP, содержащий пакеты и классы Java, а также ресурсы проекта

## Классы-оболочки. Основные понятия, назначение. Упаковка, распаковка, сравнение объектов. Примеры классов-оболочек.

Обертка — это специальный класс, который хранит внутри себя значение примитива.

1. примитивы не имеют методов. В обертках можем написать. Например, у примитивных типов нет метода toString(), поэтому ты не сможешь, например, преобразовать число int в строку
2. Объекты классов-обертки являются неизменяемыми (Immutable).
3. от классов-обертки будет объект. Примитивы не являются объектами.

Wrapper Classes for Primitive Data Types			
Primitive Data Types	Wrapper Classes	Primitive Data Types	Wrapper Classes
int	Integer	float	Float
short	Short	double	Double
long	Long	char	Character
byte	Byte	boolean	Boolean

Особенности примитивов и их классов-обертки: автоупаковка/автораспаковка (Autoboxing/Autounboxing). Не работают для массивов. Параметры также подлежат им.

**Автоупаковка (autoboxing)**— процесс, когда переменной класса-обертки можно присваивать значение примитивного типа. Этот процесс называется автоупаковкой

**Автораспаковкой (autounboxing)** – процесс, когда переменной примитивного типа можно присваивать объект класса-обертки.

Примитивы:

- имеют преимущество в производительности

Обертки:

- Позволяют не нарушать принцип “все является объектом”, благодаря чему числа, символы и булевы значения true/false не выпадают из этой концепции
- Расширяют возможности работы с этими значениями, предоставляя удобные методы и поля
- Необходимы, когда какой-то метод может работать исключительно с объектами

Несмотря на то, что значения базовых типов могут быть присвоены объектам классов-оболочек, сравнение объектов между собой происходит по ссылкам.

Метод equals() сравнивает не значения объектных ссылок, а значения объектов, на которые установлены эти ссылки. Поэтому вызов `oa.equals(ob)` возвращает значение true.

Значение базового типа может быть передано в метод equals(). Однако ссылка на базовый тип не может вызывать методы.

## Javadoc. Синтаксис и основные теги.

Требования к документам:

1. Не документировать очевидные вещи (setter'ы и getter'ы, циклы по массивам и листам, вывод логов и прочее)
2. Поддерживать документацию в актуальном состоянии
3. Описывать входящие параметры если нужно

Javadoc-комментарий (он может включать в себя HTML тэги и специальные javadoc тэги, которые позволяют включать дополнительную информацию и ссылки)

Структура каждого javadoc-комментария такова:

- первая строчка, которая попадает в краткое описание класса (отделяется точкой и пустой строкой);
- основной текст, который вместе с HTML тэгами копируется в основную документацию;
- входящие параметры (если есть);
- выбрасываемые исключения (если есть);
- возвращаемое значение (если есть);
- служебные javadoc-тэги.

Типы тегов:

- Блочные теги: начинается с @tag и оканчивается с началом следующего тега: @param x a value
- Строчные теги: ограничены фигурными скобками. Могут встречаться в теле других тегов: Use a {@link java.lang.Math#log} for positive numbers.

Тег	Синтаксис	Пример
Описывает параметров методов и конструкторов	@param <имя параметра> <описание>	@param x a value
Описывает возвращаемое значение метода	@return <описание>	@return the factorial of <code>x</code>
Ссылка на дополнительную информацию	@see <имя класса> @see [<имя класса>]#<имя члена> @see "<Текст ссылки>"	@see Math#log10 @see "The Java Programming language Specification, p. 142"
Текущая версия класса/пакета	@version <описание версии> @version <описание версии>	@version 5.0
Помечает возможности, которые не следует использовать	@deprecated <комментарий>	@deprecated replaced by {@link #setVisible}
Описывает автора класса/пакета	@author <имя автора>	@author Ivanov
Ссылка на другую сущность	{@link <класс>#<член><текст>}	{@link java.lang.Math#Log10 Decimal Logarithm} {@link Math}

		{@link Math#Log10} {@link #factorial()} calculates factorial}
Заменяется на значение поля	{@value <имя класса>#<имя поля>}	Default value is {@value #DEFAULT_TIME}
Предназначен для вставки фрагментов кода. Внутри тэга HTML не распознается	{@code <код>}	Is equivalent of {@code Math.max(a, b)}.

Для компиляции JavaDoc используется инструмент Javadoc

Применение: javadoc <опции> <список пакетов> <список файлов>

## 6. Потоки данных в Java. Основные виды и классы.

В JAVA существует 2 типа потоков данных:

- Символьные потоки (text-streams, последовательности 16-битовых символов Unicode), содержащие символы.



- Байтовые потоки (binary-streams), содержащие восьми битную информацию.



Классы разделяются также по направлению потоков:

- Потоки ввода (*input*)
- Потоки вывода (*output*)

Потоки связаны с некоторым источником данных:

- файл на диске
- сокет (при передаче данных по сети)
- устройство
- буфер в памяти

Различные потоки могут поддерживать передачу различных данных:

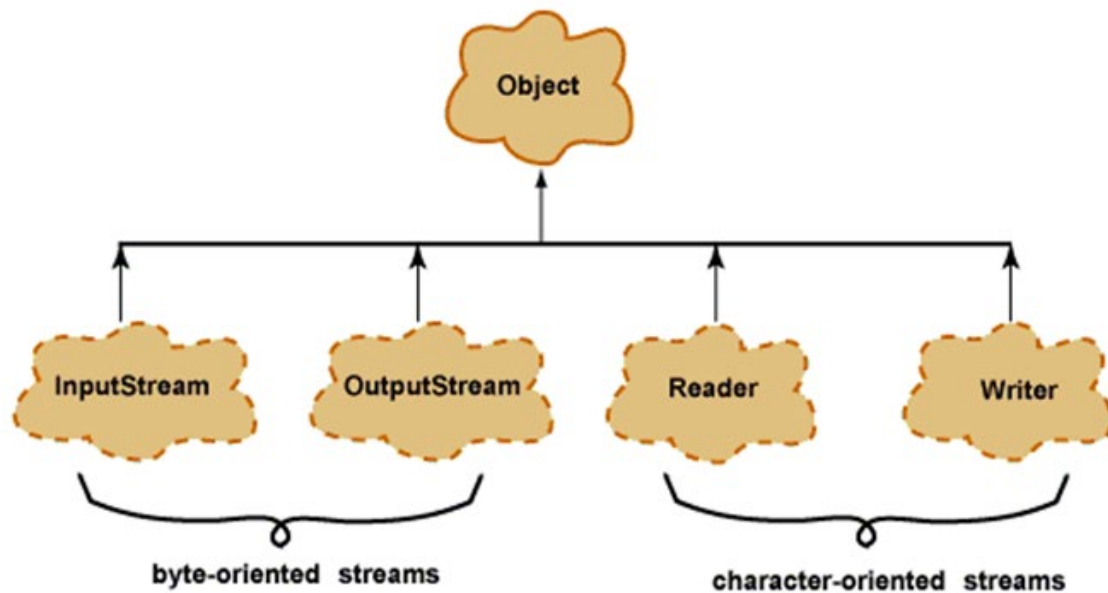
- байты -
- символы
- значения примитивных типов
- объекты

Байтовые потоки (byte streams) используются для передачи данных в виде последовательности байт. `java.io.InputStream` – абстрактный класс, управляющий байтовым потоком ввода. `java.io.OutputStream` – абстрактный класс, управляющий байтовым потоком вывода.

Буферизованные потоки (buffered streams) используют буфер для промежуточного хранения данных, повышают производительность ввода/вывода за счет уменьшения

количества обращений к источнику данных, выступают в качестве оболочек для небуферизованных потоков

В Java определены четыре основных абстрактных классов для работы с потоками:



Общая схема работы с потоками в Java описывается тремя простыми шагами:

1. Создать потоковый объект и ассоциировать его с файлом на диске.
2. Пока есть информация, читать/писать очередные данные в/из потока.
3. Закрыть поток.

## 7. Что такое Code Convention. Что содержит Code Convention для Java.

Code Convention (стандарт кодирования) – набор правил и соглашений, используемых при написании исходного кода на некотором языке программирования. Наличие общего стиля программирования облегчает понимание и поддержание исходного кода, написанного более чем одним программистом, а также упрощает взаимодействие нескольких человек при разработке программного обеспечения.

Обычно, стандарт оформления кода описывает:

- способы выбора названий и используемый регистр символов для имён переменных и других идентификаторов:
  - о запись типа переменной в её идентификаторе (венгерская нотация) и
  - о регистр символов (нижний, верхний, «верблюжий», «верблюжий» с малой буквы), использование знаков подчёркивания для разделения слов;
- стиль отступов при оформлении логических блоков — используются ли символы табуляции, ширина отступа;
- способ расстановки скобок, ограничивающих логические блоки;
- использование пробелов при оформлении логических и арифметических выражений;
- стиль комментариев и использование документирующих комментариев.

· ограничение размера кода по горизонтали (чтобы помещался на экране) и вертикали (чтобы весь код файла держался в памяти), а также функции или метода в размер одного экрана.

### **Code Convention for Java:**

Имена файлов Java должны совпадать с именем класса, описанного в нем. Единственно допустимое расширение файла, содержащего Java код - .java.

Скомпилированный байт-код - .class

ReadMe файл в репозитории должен иметь имя README.md.

Файл лицензии должен иметь имя LICENSE.md.

Файл .gitignore должен иметь имя .gitignore.

**Следует избегать строк длиннее чем 80 символов и разбивать их на несколько. Отступ 4 пробела, а не табуляция.**

**В большинстве случаев код должен быть самодокументированным, комментарии следует добавлять только при необходимости.**

**Отдельные комментарии предшествуют коду и имеют такой же отступ. Начинаются с заглавной буквы.**

**Комментарии в конце строки должны быть однострочными и имеют отступ в 1 пробел от кода. Начинаются с прописной буквы.**

**Javadoc комментарии стоит указывать для public элементов и для нетривиальных protected, package, private элементов.**

**Следует разбивать текст на параграфы с помощью тега <p>...</p>. Параграфы отделяются друг от друга пустой строкой.**

**Предпочтительнее писать одно объявление в строке.**

**После указания типа данных следует 1 пробел и имя переменной.**

**Обязательно один тип данных в строке.**

**Для указания массива квадратные скобки помещаются после имени.**

**Инициализация по возможности должна происходить при объявлении переменных.**

**Для интерфейсов выбирается имя без префиксов или суффиксов.**

**Реализации интерфейса имеют суффикс - имя интерфейса**

## **8. Что такое логирование. Библиотека log4j. Что такое Logger, Appender и Layout.**

**Логирование** – ведение протокола, или протоколирование, т.е хронологическая запись с различной (настраиваемой) степенью детализации сведений о происходящих в системе событиях (ошибки, предупреждения, сообщения), обычно в файл.

Логи предназначены, как правило, для разработчиков чтобы:



- определить, что же делает система прямо сейчас, не прибегая к помощи отладчика, т.к. это иногда не оправдано;
- провести «расследование» обстоятельств, которые привели к определённому состоянию системы (например, падению или багу);
- проанализировать, на что тратится больше времени/ресурсов, т.е. профилирование.

Log4j – фреймворк для упрощения реализации рутинных операций по логированию некоторых событий, которые происходят во время работы Вашего приложения, написанного на Java. Что значит рутинных – при отслеживании работы приложения, т.е. трассировке, выполнения конструкторов, методов, блоков обработки критических ситуаций, разработчики сталкиваются с одними и теми же операциями, а именно:

- Выбор хранилища – консоль, файл, СУБД;
- Конфигурация хранилища – именование хранилища, объем хранилища, путь к хранилищу, сохранение конфигурации;
- Форматирование записей журнала – дата/время, класс/метод и т.д., гибкое форматирование.

Все эти операции фреймворк позволяет автоматизировать, а конфигурацию параметров хранить в одном файле конфигураций отдельно.

Любой регистратор событий состоит из трех элементов:

- собственно регистрирующего - logger - позволяет инициализировать запись в лог тех или иных сообщений;
- направляющего вывод - appender - указывают возможные места назначения выводимого в лог сообщения;
- форматирующего вывод - layout.

**Регистрирующий элемент** представляет собой создаваемый программистом объект класса Logger или LogManager, позволяющий инициализировать запись в лог тех или иных сообщений:

```
Logger log = LogManager.getLogger(Two.class);
```

Уровни логирования в log4j:

- FATAL - произошла фатальная ошибка - у этого сообщения наивысший приоритет
- ERROR - в программе произошла ошибка
- WARN - предупреждение в программе что-то не так
- INFO - информация.
- DEBUG - детальная информация для отладки
- TRACE– трассировка всех сообщений в указанный аппендер



OFF< TRACE< DEBUG< INFO< WARN< ERROR< FATAL< ALL

Такая иерархия означает что если установлен уровень логирования DEBUG, то лог будет содержать и все вышестоящие уровни (INFO, WARN, ERROR, FATAL).

Данный объект для поддержки указанных уровней имеет соответствующие методы .trace(), .debug(), .info(), .warn(), .error(), .fatal() которые передают в лог сообщения с соответствующим уровнем.

**Appender'ы** указывают возможные места назначения выводимого в лог сообщения: файл, консоль и т. д. Каждому из них соответствует класс, реализующий интерфейс org.apache.log4j.Appender. Кроме того, вывод в базу данных можно произвести с помощью класса JDBCAppender, в журнал событий ОС - NTEventLogAppender, на SMTP-сервер - SMTPAppender.

Если логгер - это та точка, откуда уходят сообщения в коде, то аппендер - это та точка, куда они приходят в конечном итоге. Например, файл, консоль, база данных, SMTP, Telnet, SysLog, Сокет и др.

**Appender'ы** указывают возможные места назначения выводимого в лог сообщения: файл, консоль и т. д. Каждому из них соответствует класс, реализующий интерфейс org.apache.log4j.Appender. Кроме того, вывод в базу данных можно произвести с помощью класса JDBCAppender, в журнал событий ОС - NTEventLogAppender, на SMTP-сервер - SMTPAppender.

Если логгер - это та точка, откуда уходят сообщения в коде, то аппендер - это та точка, куда они приходят в конечном итоге. Например, файл, консоль, база данных, SMTP, Telnet, SysLog, Сокет и др.

**Формат вывода** записи лога определяется параметрами классов, производных от родительского Layout. Все методы класса Layout предназначены только для создания подклассов.

Наиболее распространенные виды форматов:

- org.apache.log4j.SimpleLayout - наиболее простой вариант. На выходе отображается уровень вывода и сообщение.
- org.apache.log4j.HTMLLayout - данный компоновщик форматирует сообщения в виде HTML-страницы.
- org.apache.log4j.xml.XMLLayout - формирует сообщения в виде XML.

## 9. Что такое класс. Что такое объект. Зачем нужно ООП. Синтаксис и пример определения класса в Java. Вложенные классы.

**Класс** – это некоторая модель еще не существующих сущностей (объектов), описывающая их возможное состояние и поведение. Состояние реализуется посредством полей класса, представляющих собой переменные, а поведение посредством методов, представляющих собой функции.

**Объект** – это отдельный представитель (экземпляр) класса, имеющий конкретное состояние и реализующий своё поведение, полностью определяемое классом

\*Элементы класса (class members):

\*Поля (fields)

\*Методы (methods)

\*Спецификаторы доступа:

\*public – элемент класса является общедоступным

\*private – элемент доступен только методам класса

\*protected – элемент доступен только методам класса и дочерних классов

\*(не задан) – package-private – элемент доступен внутри пакета

\*Рекомендуется скрывать поля класса от внешнего доступа

## 10. Основные принципы ООП. Наследование, полиморфизм. Примеры на языке Java.

\*Наследование (inheritance)

\*У класса может быть один родитель и любое количество дочерних классов

\*Прародителем всех классов Java является класс java.lang.Object

\*Дочернему классу передаются поля и методы родительского класса

\*Дочерний класс может обращаться полям и методам родительского класса, которые:

\*объявлены со спецификатором public или protected

\*объявлены без спецификатора (package private), при условии что дочерний класс находится в одном пакете вместе с родительским

\*Дочерний класс может иметь свои собственные поля и методы, а также переопределять методы родительского класса

\*Методы родительского класса могут быть переопределены в дочернем классе

\*Для обращения к переопределенным элементам родительского класса из дочернего класса используется ключевое слово super

\*Скрытые (private) методы не могут быть переопределены

\*При переопределении методов нельзя уменьшать их видимость

\*Метод, который вызывает родительский класс, может быть переопределен

**Полиморфизм** – способность дочерних классов переопределять методы родительского класса с тем же названием, реализуя отличающееся поведение. Таким образом, иерархия классов предоставляет одинаковый интерфейс, но разные реализации одного и того же поведения для разных классов.

## 11. Инкапсуляция. Спецификаторы доступа в Java.

**Инкапсуляция** – механизм задания уровня доступа к элементам класса. Инкапсуляция предохраняет данные объекта от нежелательного доступа, позволяя объекту самому управлять доступом к своим данным.

Открытые члены класса составляют внешнюю функциональность, которая доступна другим объектам. Закрытыми (`private`) обычно объявляются независимые от внешнего функционала члены, а также вспомогательные методы, которые являются лишь деталями реализации и неуниверсальны по своей сути. Благодаря сокрытию реализации класса можно менять внутреннюю логику отдельного класса, не меняя код остальных компонентов системы.

\*Спецификаторы доступа:

\*`public` – элемент класса является общедоступным

\*`private` – элемент доступен только методам класса

\*`protected` – элемент доступен только методам класса и дочерних классов

\*(не задан) – `package-private` – элемент доступен внутри пакета

\*Спецификатор `final`, в объявлении класса запрещает наследование от данного класса

\*Спецификатор `final` в объявлении метода запрещает переопределение данного метода при наследовании

\*методы, вызываемые в конструкторе, рекомендуется объявлять со спецификатором `final`

Преимущества инкапсуляции

- Поля класса можно сделать только для чтения или только для записи.
- Класс может иметь полный контроль над тем, что хранится в его полях.

## 12. Абстрактные классы и интерфейсы (назначение, различие, использование). Основные методы класса `Object` в Java.

Абстрактный класс (`abstract class`)

\*определяет общее поведение для порожденных им классов

\*предполагает наличие дочерних классов

\*объявляется со спецификатором `abstract`

\*не может иметь объектов

\*может содержать или не содержать абстрактные методы

Класс должен быть объявлен как абстрактный если:

\*класс содержит абстрактные методы

\*класс наследуется от абстрактного класса, но не реализует абстрактные методы

\*класс имплементирует интерфейс, но не реализует все методы интерфейса

\*Абстрактный метод (`abstract method`)

\*не имеет реализации

\*объявляется со спецификатором `abstract`

\*переопределяется в дочерних классах

Абстрактные классы

\*описывают поведение для иерархии классов

\*могут обладать состоянием

- \*могут реализовывать алгоритмы
- \*объектные ссылки поддерживают динамический полиморфизм
- \*могут содержать скрытые и защищенные элементы
- \*класс может наследоваться только от одного абстрактного класса

#### Интерфейсы

- \*описывают поведение для группы классов, реализующих данный интерфейс
- \*не могут обладать состоянием
- \*(Java SE 7) не могут реализовывать алгоритмы; (Java SE 8) могут реализовывать алгоритмы по умолчанию
- \*интерфейсные ссылки поддерживают динамический полиморфизм
- \*содержат только публичные элементы
- \*класс может реализовывать несколько интерфейсов

**Интерфейсы** применяются для добавления к классам новых возможностей, которых нет и не может быть в базовых классах. Интерфейсы говорят о том, что класс может делать, но не говорят, как он должен это делать. Интерфейс только гарантирует, что класс выполняет какие-то функции, а как он их выполняет – дело неинтерфейсное

#### Стандартные интерфейсы Java

- \*java.lang.Appendable
- \*java.lang.AutoClosable
- \*java.lang.CharSequence
- \*java.lang.Cloneable
- \*java.lang.Comparable
- \*java.lang.Iterable
- \*java.lang.Readable
- \*java.lang.Runnable
- \*java.io.Serializable
- \*java.io.DataInput
- \*java.io.DataOutput

### 13. Коллекции в Java. Основные коллекции и их методы.

Коллекции – это хранилища, поддерживающие различные способы накопления и упорядочения объектов с целью обеспечения возможностей эффективного доступа к ним.

Применение коллекций обуславливается возросшими объемами обрабатываемой информации.

Коллекции в языке Java объединены в библиотеке классов java.util и представляют собой контейнеры, т.е объекты, которые группируют несколько элементов в отдельный модуль.

Коллекции используются для хранения, поиска, манипулирования и передачи данных.

Коллекции – это динамические массивы, связанные списки, деревья, множества, хэш-таблицы, стеки, очереди.

Коллекции (collections) или контейнеры (containers) предназначены для работы с группой элементов

- \*Элементом коллекции является объект
- \*Коллекции обеспечивают хранение элементов и доступ к ним

Во-первых, в основе всех коллекций лежит применение того или иного интерфейса, который определяет базовый функционал. Среди этих интерфейсов можно выделить следующие:

- **Collection**: базовый интерфейс для всех коллекций и других интерфейсов коллекций
- **Queue**: наследует интерфейс **Collection** и представляет функционал для структур данных в виде очереди
- **Deque**: наследует интерфейс **Queue** и представляет функционал для двунаправленных очередей
- **List**: наследует интерфейс **Collection** и представляет функциональность простых списков
- **Set**: также расширяет интерфейс **Collection** и используется для хранения множеств уникальных объектов
- **SortedSet**: расширяет интерфейс **Set** для создания сортированных коллекций
- **NavigableSet**: расширяет интерфейс **SortedSet** для создания коллекций, в которых можно осуществлять поиск по соответствию
- **Map**: предназначен для созданий структур данных в виде словаря, где каждый элемент имеет определенный ключ и значение. В отличие от других интерфейсов коллекций не наследуется от интерфейса **Collection**

Среди методов интерфейса **Collection** можно выделить следующие:

- **boolean add (E item)**: добавляет в коллекцию объект *item*. При удачном добавлении возвращает **true**, при неудачном - **false**
- **boolean addAll (Collection<? extends E> col)**: добавляет в коллекцию все элементы из коллекции *col*. При удачном добавлении возвращает **true**, при неудачном - **false**
- **void clear ()**: удаляет все элементы из коллекции
- **boolean contains (Object item)**: возвращает **true**, если объект *item* содержится в коллекции, иначе возвращает **false**
- **boolean isEmpty ()**: возвращает **true**, если коллекция пуста, иначе возвращает **false**
- **Iterator<E> iterator ()**: возвращает объект **Iterator** для обхода элементов коллекции
- **boolean remove (Object item)**: возвращает **true**, если объект *item* удачно удален из коллекции, иначе возвращается **false**
- **boolean removeAll (Collection<?> col)**: удаляет все объекты коллекции *col* из текущей коллекции. Если текущая коллекция изменилась, возвращает **true**, иначе возвращается **false**
- **boolean retainAll (Collection<?> col)**: удаляет все объекты из текущей коллекции, кроме тех, которые содержатся в коллекции *col*. Если текущая коллекция после удаления изменилась, возвращает **true**, иначе возвращается **false**
- **int size ()**: возвращает число элементов в коллекции
- **Object[] toArray ()**: возвращает массив, содержащий все элементы коллекции

## **14. Регулярные выражения в Java. Синтаксис и основные метасимволы. Классы Java для работы с регулярными выражениями и их основные методы.**

Регулярные выражения представляют собой формальный язык поиска и редактирования подстрок в тексте. Допустим, нужно проверить на валидность e-mail адрес. Это проверка на наличие имени адреса, символа **@**, домена, точки после него и доменной зоны.

Вот самая простая регулярка для такой проверки:

```
^[A-Z0-9+_.-]+@[A-Z0-9.-]+$
```

В коде регулярные выражения обычно обозначается как `regex`, `regexpr` или `RE`.

Спецификаторы:

Символы могут быть буквами, цифрами и метасимволами, которые задают шаблон:

Есть и другие конструкции, с помощью которых можно сокращать регулярки:

- `\d` — соответствует любой одной цифре и заменяет собой выражение `[0-9]`;
- `\D` — исключает все цифры и заменяет `[^0-9]`;
- `\w` — заменяет любую цифру, букву, а также знак нижнего подчёркивания;
- `\W` — любой символ кроме латиницы, цифр или нижнего подчёркивания;
- `\s` — поиск символов пробела;
- `\S` — поиск любого непробельного символа.

Квантификаторы:

Это специальные ограничители, с помощью которых определяется частота появления элемента — символа, группы символов, etc:

`?` — делает символ необязательным, означает 0 или 1. То же самое, что и `{0,1}`.

`*` — 0 или более, `{0,}`.

`+` — 1 или более, `{1,}`.

`{n}` — означает число в фигурных скобках.

`{n,m}` — не менее `n` и не более `m` раз.

`*?` — символ `?` после квантификатора делает его ленивым, чтобы найти наименьшее количество совпадений.

Спец. символ	Зачем нужен
<code>.</code>	Задаёт <b>один</b> произвольный символ
<code>[]</code>	Заменяет символ из квадратных скобок
<code>-</code>	Задаёт один символ, которого не должно быть в скобках
<code>[^]</code>	Задаёт один символ из <b>не</b> содержащихся в квадратных скобках
<code>^</code>	Обозначает начало последовательности
<code>\$</code>	Обозначает окончание строки
<code>*</code>	Обозначает произвольное число повторений одного символа
<code>?</code>	Обозначает строго одно повторение символа
<code>+</code>	Обозначает один символ, который повторяется несколько раз
<code> </code>	Логическое <b>ИЛИ</b> . Либо выражение до, либо выражение после символа
<code>\</code>	Экранирование. Для использования метасимволов в качестве обычных
<code>()</code>	Группирует символы внутри
<code>{ }</code>	Указывается число повторений предыдущего символа

Квантификатор	Число повторений	Пример	Подходящие строки
<code>{n}</code>	Ровно <code>n</code> раз	<code>Ха{3}ха</code>	Хаааха
<code>{m,n}</code>	От <code>m</code> до <code>n</code> включительно	<code>Ха{2,4}ха</code>	Хаа, Хааа, Хааааха
<code>{m,}</code>	Не менее <code>m</code>	<code>Ха{2,}ха</code>	Хааха, Хаааха, Хааааха и т. д.
<code>{,n}</code>	Не более <code>n</code>	<code>Ха{,3}ха</code>	Хха, Хаха, Хааха, Хаааха



Обратите внимание, что квантификатор применяется только к символу, который стоит перед ним. и есть три режима: жадный (поиск самого длинного совпадения), сверхжадный (как первый, но без реверсивного поиска при захвате строки) , ленивый (поиск самого короткого совпадения)

### Особенности регулярных выражений в Java:

Для, например, метасимволов, используется двойная косая черта, чтобы указать компилятору Java, что это элемент регулярки.

Java RegExр обеспечиваются пакетом [java.util.regex](#). Здесь ключевыми являются три класса:

1. `Matcher` — выполняет операцию сопоставления в результате интерпретации шаблона.
2. `Pattern` — предоставляет скомпилированное представление регулярного выражения.
3. `PatternSyntaxException` — предоставляет непроверенное исключение, что указывает на синтаксическую ошибку, допущенную в шаблоне RegEx.

Также есть интерфейс `MatchResult`, который представляет результат операции сопоставления

Пример:

```
List emails = new ArrayList();
emails.add("name@gmail.com");
//Неправильный имейл:
emails.add("@gmail.com");
String regex = "[A-Za-z0-9+_.-]+@(.+)$";
Pattern pattern = Pattern.compile(regex);
for(String email : emails){
    Matcher matcher = pattern.matcher(email);
    System.out.println(email + " : " + matcher.matches());}
```

## 15. Работа с xml в Java. Технологии DOM и SAX. Их назначение, преимущества и отличия.

XML — это описанная в текстовом формате иерархическая структура, предназначенная для хранения любых данных. Визуально структура может быть представлена как дерево элементов. Элементы XML описываются тегами.

Первая строка XML-документа называется объявление XML (англ. XML declaration) — это строка, указывающая версию XML. В версии 1.0 объявление XML может быть опущено, в версии 1.1 оно обязательно. Также здесь может быть указана кодировка символов и наличие внешних зависимостей

Важнейшее обязательное синтаксическое требование заключается в том, что документ имеет только один корневой элемент (англ. root element).

Существуют две стратегии обработки XML документов: DOM (Document Object Model) и SAX (Simple API for XML).

Основное их отличие связано с тем, что использование DOM позволяет читать и вносить изменения в существующий XML-документ, а также создавать новый.

Стратегия использования SAX основывается на том, что содержимое XML-документа только анализируется.



XML-текст может быть больших размеров: DOM должен весь документ «заглотить» и проанализировать, а SAX-парсер обрабатывает XML-документ последовательно и не требует дополнительной памяти.

Обработка в SAX включает в себя следующие шаги:

1. Создание обработчика событий.
2. Создание парсера SAX.
3. Назначение обработчика событий для парсера.
4. Разбор документа с посылкой каждого события в обработчик.

Объект Document является интерфейсом, так что его экземпляр не может быть создан непосредственно, обычно вместо этого приложение использует фабрику. Подробности этого процесса различаются от реализации к реализации, но идеи одни и те же. (Опять-таки, Уровень 3 стандартизирует эту задачу.) Например, в среде Java разбор файла является 3-шаговым процессом:

1. Создание DocumentBuilderFactory. Этот объект создает DocumentBuilder.
2. Создание DocumentBuilder. DocumentBuilder действительно выполняет разбор для создания объекта Document.
3. Разбор файла для создания объекта Document.

Использовать вам DOM или SAX, зависит от нескольких факторов:

- Назначение приложения: Если вы собираетесь делать изменения в данных и выводить их как XML, то в большинстве случаев способом для этого является DOM. Нельзя сказать, что вы не можете делать изменения при использовании SAX, но этот процесс значительно более сложен, так как вы должны делать изменения в копии данных, а не в самих данных.
- Объем данных: Для больших файлов SAX является лучшим выбором.
- Как будут использованы данные: Если на самом деле будет использована только небольшая часть данных, вам, возможно, лучше применить SAX, чтобы выделить ее в ваше приложение. С другой стороны, если вы знаете, что вам придется ссылаться назад в большом объеме информации, которая уже была обработана, SAX, возможно, не является правильным выбором.
- Требования к быстродействию: реализации SAX обычно быстрее, чем реализации DOM.

Важно помнить, что SAX и DOM не являются взаимоисключающими. Вы можете использовать DOM для создания потока событий SAX, и вы можете использовать SAX для создания дерева DOM. Фактически, большинство парсеров, применяемых для создания дерева DOM, используют SAX, чтобы сделать это!

## 16. Что такое файл \*.properties. Работа с базой данных в Java. Что такое JDBC. Основные интерфейсы JDBC и их методы.

**файл \*.properties** - файлы свойств предназначены для того, чтобы хранить в них какие-то статические данные, необходимые для проекта, например конфигурационные параметры программы.

Данные в этих файлах представлены в виде «**ключ** =|: **значение**», где

ключ – это уникальное имя, по которому можно получить доступ к значению, хранимому под этим ключом.

значение – это текст, либо число, которое вам необходимо для выполнения определённой логики в программе.

Разделителем могут выступать знаки «=» или «:».

\*.properties файлы могут использовать знак решетки (#) или восклицательный знак (!) как первый, не пустой символ в строке для обозначения последующего текста в качестве комментария.

Для чтения и разбора файла \*.properties используется объект класса java.util.Properties. Метод load() считывает файл, а метод getProperty("свойство") возвращает значение указанного свойства.

Для того, чтобы получить доступ к базе данных, Java позволяет использовать JDBC (Java Database Connectivity) API, который входит в стандартную библиотеку Java. JDBC позволяет подключиться к любой базе данных: Postgres, MySQL, SQL Server, Oracle и т. д. – при наличии соответствующей реализации драйвера, необходимого для подключения. JDBC – это стандарт взаимодействия приложения с различными СУБД. Он основан на концепции драйверов, позволяющей получать соединение с БД по специальному URL.

Из определения выходит, что для соединения с СУБД нужен драйвер, который можно скачать на сайте производителя СУБД.

Все основные сущности в JDBC API, с которыми наиболее часто приходится работать, являются интерфейсами:



Connection.

- **Statement.** - Данный объект используется для выполнения простых SQL-запросов без параметров. Содержит базовые методы для выполнения запросов и извлечения результатов.
- **PreparedStatement.** - и используется для выполнения SQL-запросов с или без входных параметров. Добавляет методы управления входными параметрами. Также часто используется для выполнения множества похожих запросов.
- **CallableStatement.** -и используется для вызовов хранимых процедур. Добавляет методы для манипуляции выходными параметрами
- **ResultSet** - результирующий набор данных и обеспечивает приложению построчный доступ к результатам запросов. При обработке запроса *ResultSet* поддерживает указатель на текущую обрабатываемую запись.

DatabaseMetaData - Позволяет получить метаинформацию о схеме базы данных, а именно какие в базе данных есть объекты - таблицы, колонки, индексы, триггеры, процедуры и так далее.

JDBC драйвер конкретной СУБД как раз и предоставляет реализации этих интерфейсов

## **17. Коллективная разработка программного продукта. Задачи, функции и условия эффективной работы команды.**

Для успешного завершения необходимо решить 3 основных задачи:

- Как организовать людей?
- Как организовать процесс проектирования?
- Как должно быть устроено приложение? (грамотное техническое задание)

Выделяются 6 ролевых групп в команде:

- Управление продуктом (менеджеры, представители заказчика) - т.е. как продукт должен выглядеть и что должен делать
- Управление программой - разработка основных спецификаций программного продукта
- Разработчики
- Тестирование
- Обучение - подготовка документации и разработка инструкций для конечных пользователей
- Логистика

Условие работы команды:

- Четкое понимание своей роли в команде, что позволяет выполнять свои задачи не пересекаясь с другими
- Спецификации и сроки должны быть согласованы со всеми членами команды
- Члены команды хорошо взаимодействуют друг с другом и используют взаимное уважение и профессиональные качества друг друга
- Все члены команды имеют четкое представление о модели процесса которое будет использоваться в ходе выполнения проекта
- Каждый участник знает все аспекты разработки

## **18. Коллективная разработка программного продукта. Этапы подбора команды. Качество программного продукта.**

- Анализ требуемых ресурсов - определить набор требований к квалификации и проф.качествам персонала - подбор ТЗ, выбор менеджера проекта
- Анализ имеющихся ресурсов - сопоставить то что есть тому что нужно
- Выборка решений для заполнения ролей - заключение контрактов или перераспределение ресурсов
- Планирование работы команды

- Координация действий по обучению
- Обучение сотрудников

Качество программного продукта определяется:

- Степенью соответствия системным требованиям
- Соответствие спецификациям заложенным в ТЗ
- Соответствие ожиданиям пользователя
- Своевременность выполнения проекта

Характеристики качества:

- Удовлетворяет требованиям заказчика
- Соответствует спецификациям и ограничениям проекта
- Все проблемы и аспекты важные для бизнеса заказчика и пользователя выявлены до того как продукт был внедрен заказчиком
- Гарантируется что конечный пользователь знает как применить программный продукт
- Гарантируется плавное внедрение продукта в бизнес заказчика

## **19. Системы управления исходным кодом. Возможности, цели, словарь.**

Управление версиями – ключевой аспект любого проекта разработки ПО. Хорошая система управления исходным кодом вписывается в цикл разработки и является естественным расширением инструментария разработчика.

Система управления версиями предоставляет следующие возможности:

1. Поддержка файлов в репозитории
2. Поддержка проверки файлов в репозитории
3. Нахождение конфликтов при изменении исходного кода и обеспечение синхронизации при работе в многопользовательской среде разработки
4. Отслеживать авторов изменений
5. Возможность управления конфигурацией файлов для совместимых и повторяющихся сборок

Контроль версий служит нескольким основным целям:

1. Позволяет безопасно хранить успешные версии исходников, позволяя вернуться к подходящей стабильной версии при возникновении неполадок.
2. Помогает членам команды работать одновременно над исходниками проекта так, чтобы не портить друг другу код. Это можно сделать двумя способами:
  - 1) Заблокировать файл, когда над ним работает пользователь. Такая модель называется «Блокировка, изменение, разблокировка».
  - 2) Позволяет разработчикам редактировать один и тот же файл одновременно, а затем сливать изменения воедино в новую версию, содержащую изменения всех разработчиков. Такая модель называется «Копирование, изменение, слияние».

Словарь системы управления версиями:

1. Репозиторий – это центральное место, где хранятся файлы.
2. Выгрузка (Check-out) – извлечение файлов из репозитория в рабочую директорию вашей локальной системы.
3. Фиксация изменений (Commit) – синхронизация локальных файлов с файлами репозитория.
4. Конфликт – невозможность объединения изменений в файле

5. Обновление файлов – синхронизация файлов репозитория с локальной рабочей версией
6. Ревизия – порядковый номер сборки после фиксации изменения.
7. Персональная ветка – позволяет разработчикам работать вне главного проекта.
8. Метка – позволяет пометить начало отсчета изменений, а также сгруппировать файлы, пригодные для использования в блоках.

## 20. Архитектура систем управления версиями. СУВ CVS и SVN.

# Системы управления версиями

SVN, Git:

1. Хранение всех версий программного продукта
2. Независимость от аппаратной части
3. Коллективная разработка ПО

Архитектуры:


- По способу хранения изменений
  - модель моментальных снимков
  - модель хранения изменений (сохраняются только измененные файлы)
- По способу хранения данных
  - Централизованные - есть сервер через который происходят потоки изменения всех разработчиков.
  - Распределенные (GIT)

RefLinks - если файл не изменялся, то ссылка на него из прошлой версии.

CVS (Control version system) – одна из первых систем управления версиями, появилась в 80х годах. Использует централизованную архитектуру. CVS поддерживает минимальный набор операций (импорт, экспорт, создание рабочей копии проекта, поддержку меток и веток).

Недостатки:

1. Медленные операции по метке и ветвления.
2. Невозможно переименование файлов и перемещение каталогов.
3. Commit`ы не атомарны.
4. Проблема с поддержкой бинарных файлов.

SubVersion (SVN) поддерживает  атомарные коммиты, быстрое создание ветвей и меток, переименование и перемещение файлов и директорий, поддерживает бинарные файлы, сетевые транзакции и прочее. Также достоинством является легкость настройки сервера для предоставления доступа по HTTP к репозиторию. SVN предоставляет более гибкую модель аутентификации и управления правами пользователей. В отличие от CVS, которая ведет отдельную историю версий для каждого файла SVN отслеживает целые ревизии. Именно введение ревизий позволило реализовать атомарность обновлений.

## 21. Архитектура систем управления версиями. СУВ Mercurial и Bazaar.

Архитектуры:

- По способу хранения изменений
  - модель моментальных снимков
  - модель хранения изменений (сохраняются только измененные файлы)
- По способу хранения данных
  - Централизованные - есть сервер через который проходят потоки изменения всех разработчиков.
  - Распределенные (GIT)

Mercurial основан на философии распределенных хранилищ без выделенного центрального сервера. Это основное достоинство и основной недостаток распределенных систем.

Вместо использования центрального репозитория каждый разработчик имеет полную копию хранилища на локальной машине. Если разработчик работает над несколькими ветками, то у него будет несколько полных копий репозитория. Это устраняет зависимость от центрального сервера и потенциально делает систему более устойчивой. Разработчики должны сами продумывать и решать как и куда отсылать изменения в коде.

Для крупных проектов распределенная модель может привести к проблемам с объемам дискового пространства и падением производительности.

В Mercurial как и в SVN используется понятие ревизии. Эта система обладает всеми достоинствами что и SVN. К проектам Mercurial можно получить доступ по протоколу HTTP. Если при коммите случается конфликт Mercurial не пытается соединить 2 файла, в отличие от CVS и SVN, вместо этого он показывает конфликтующие файлы и предоставляет разработчику выбрать программу для слияния. По умолчанию Mercurial считает, что разработчик хочет получить изменения из оригинального репозитория, из которого копировался проект. В этом случае оригинальный репозиторий ведет себя как оригинальный сервер.

Bazaar использует распределенную модель и хранит локальную копию на каждой машине. Для корректного протоколирования изменений разработчику необходимо представиться. Затем необходимо создать новую ветку проекта на локальной машине, т.е. клонировать его. Система Bazaar поддерживает доступ по HTTP. Если разработчик допустил ошибку, Bazaar позволит откатить изменения при помощи команды UnCommit. Bazaar поддерживает автоматическое слияние файлов при конфликте. По разрешению конфликта Bazaar сообщит слияние локального и удаленного репозитория. Система Bazaar использует гибкие инновационные средства с хорошо продуманными возможностями. Эта система легче в использование, чем Mercurial, может использовать центральный репозиторий, что позволяет разработчикам применять лучшее из обоих подходов

## **22. Система управления версиями Git. Хранение файлов.**

Это распределенная система управления версиями файлов, которая написана на языке C. Проект был создан Линусом Торвальдсом в 2005 году для управления разработкой ядра Linux.

Особенностью Git является то, что работа над версиями проекта может происходить не в хронологическом порядке, т.е. разработка может вестись в нескольких ветках, которые могут сливаться или разделяться в любой момент времени.



Каждый файл проекта Git состоит из имени и содержания. Имя это первые 20 байтов данных, записываемое 40 символами в шестнадцатеричной системе счисления. Имя получается хешированием файла по алгоритму SHA-1.

Работу с версиями файлов Git можно сравнить обычными файловыми операциями. Структура состоит из 4 типов объектов:

- Blob – тип данных, который вмещает содержимое файла и хэш, является основным и единственным носителем данных в Git.
- Tree – тип данных, который содержит собственный хэш, хэш Blob'ов и деревьев, права доступа и символьное имя объектов.
- Commit – собственный хеш код, ссылку на одно дерево, ссылку на предыдущий Commit, имя автора и время создания Commit'а, имя коммиттера, и произвольный блок данных. Данный объект призван хранить снимок групп файлов в определённый момент времени.
- References – тип данных, содержащий ссылку на любой из 4 объектов. Основная его цель прямо или косвенно указывать на объект и являться синонимом файла, на который он ссылается.

Ветвь (Branch) – символьная ссылка, которая указывает на последний в хронологии коммит и хранит SHA-хэш объекта.

Тэг – тип данных, который в отличие от ветвей неизменно ссылается на один и тот же объект типов blob, tree, commit или tag.

Таким образом, проект в Git представляет собой набор blob'ов, которые связаны сетью деревьев. Полученная иерархическая структура может быть отражена в виде коммитов (версий), а для понимания их структуры в Git присутствуют такие объекты как ссылки. Основная идея – работа над группами файлов.

Большинство система хранит информацию как список изменений для файлов. Эти системы относятся к хранимым данным как к набору файлов и изменений, сделанных для каждого из них.

Git считает хранимые данные набором слепков небольшой файловой системы. Каждый раз когда разработчик фиксирует текущую версию проекта, Git сохраняет слепок того, как выглядят файлы проекта на текущий момент времени. Ради эффективности, если файл не менялся, Git не сохраняет его снова, а делает ссылку на предыдущую версию.

Файлы в системе Git могут находиться в одном из 3 состояний:

1. Зафиксированный – означает, что файл уже сохранен в локальной базе.
2. Измененный – к таким файлам относятся те, которые поменялись, но еще не были зафиксированы.
3. Подготовленный – это такой файл, который был изменен и отмечен для включения в следующий коммит.

## **23. Система управления версиями Git. Порядок работы с репозиториями, в том числе удаленными.**

Git это распределенная система управления версиями файлов. Особенностью Git является то, что работа над версиями проекта может происходить не в хронологическом порядке, т.е. разработка может вестись в нескольких ветках, которые могут сливаться или разделяться в любой момент времени.



INIT - создает в текущем каталоге новый подкаталог с именем .Git, содержащий все необходимые файлы репозитория. На этом этапе проект еще не находится под версионным контролем, если разработчик хочет добавить под версионный контроль существующие файлы, то их необходимо проиндексировать и осуществить первую фиксацию изменений.

ADD - Индексация файлов осуществляется этой командой.

COMMIT - Фиксация изменений

CLONE - Если разработчик желает получить копию существующего репозитория Git, то ему необходимо выполнить эту команду

Совместная работа включает в себя управление удаленными репозиториями и помещением данных тогда, когда нужно обмениваться результатами работы. Управление включает в себя добавление, управление, управления метками и ветками.

REMOTE - команда чтобы посмотреть настроенные удаленные репозитории. Она перечисляет список имен сокращений для всех уже указанных репозиториях. Применение `remote -v` отображает url (сокращенное имя).

REMOTE ADD “имя сокращения” url - выполняется чтобы добавить удалённый репозиторий под именем-сокращением.

FETCH “сокращенное имя” - команда для получения данных из удалённых репозиториях, следует выполнить команду. Связывается с удалённым репозиторием и забирает все те данные проекта, которых нет у разработчика. Команда `fetch` забирает данные в локальный репозиторий разработчика, но не сливает их с наработками рабочего каталога и не модифицирует никакие файлы, над которыми разработчик работает.

PULL url - автоматически извлекает и сливает данные из одной ветки в текущую ветку разработчика. Выполнение команды `pull` как правило извлекает данные из сервера, то есть выполняет команду `fetch` и автоматически пытается слить их с кодом, над которым работает разработчик, то есть выполнить команду **merge**.

PUSH - выполняется когда разработчик хочет поделиться своими наработками, указывая через пробел удалённый сервер и ветку, которая отправляет данные в главный центральный репозиторий. Невозможно отправить данные в центральный репозиторий, пока с него не были получены данные и не обновлён локальный репозиторий.

Чтобы изменить сокращенное имя, используемого репозитория, необходимо выполнить команду **remote rename old\_name new\_name**. Переименование репозитория также изменяет имена всех удаленных веток.

Чтобы удалить репозиторий надо выполнить **remote rm**.

## 24. Системы управления проектами. Цели и задачи.

Управление проектами - это набор технологических методов и инструментов, которые поддерживают управление проектами в организации и помогают в процессе организации работы над проектом.

То есть важно:

- Набор инструментов
- Набор методик управления

Будем понимать ПО, которое позволяет вести проект в электронной версии.

ПО для управления проектами включает в себя:

- Планирование задач (!наиболее важное)
- Составление расписания
- Контроль цен и управление бюджетом
- Распределение ресурсов
- Совместную работу
- Общение
- Быстрое управление
- Документирование
- др.

Цели использования систем УП:

- Повышение эффективности сотрудников компании при работе с проектами
- Повышение качества управления проектами
- Повышение эффективности управления всеми проектами компании

Задачи систем УП:

### 1. Планирование

- Планирование различных событий, зависящих друг от друга
- Идентификация крупных составных частей проекта (вехи проекта - milestone), их декомпозиция, создание иерархической структуры

Пример: Веха - создание дизайна. Веха\_2 : Верстка. Веха\_3: создание функционала

- Планирование расписания работ сотрудников и назначение ресурсов на конкретные задачи
- Расчет времени, необходимого на решение каждой задачи
  - планируемое время
  - трекинг времени
- Сортировка задач в зависимости от срока из завершения
- Презентация графика работ в виде диаграммы Ганта (это горизонтальная гистограмма)
- Управление несколькими проектами одновременно

### 2. Расчет критического пути

Критический путь - самый длинный путь от начала проекта до момента завершения последней задачи.

То есть это максимальный срок выполнения.

### 3. Управление данными и предоставление информации

Отображение списка задач для сотрудников. Ранние предупреждения о возможных рисках. Информация о рабочей нагрузке. Информация о ходе проекта, прогнозирование.

#### 4. Управление коммуникацией

- Обсуждение и согласование рабочих вопросов
- Фиксация проблем проекта и запросов на изменение
- Ведение рисков проекта и управление ими. Риск - это изменения в планировании, изменения в ресурсах.
- Представление в виде живой ленты событий всей коммуникации

## **25. Назначение и области применения систем управления проектами. Возможные выгоды от применения.**

### # Возможности систем управления проектами

1. Обеспечить руководителя проекта инструментарием для планирования проекта и контроля ходом реализации проекта.
2. Предоставить участникам проекта инструмент для выполнения задач и доступ ко всей необходимой информации
3. Дать руководителю подразделения дать инструмент контроля загрузки сотрудника. Предоставить информацию для принятия решения о назначении сотрудника на новые проекты или задачи.
4. Директору проектного офиса предоставить удобный инструмент для автоматизации рутинных операций и предоставить полный контроль за состоянием всех проектов компании.
5. Директору обеспечить единую панель мониторинга всех проектов компании с возможностью анализа отклонений и принятия управленческих решений.
6. Акционерам или владельцам компании видеть соответствие выполняемых проектов стратегическим целям компании.

### # Области применения систем управления проектами

1. Управление строительными проектами
2. Управление инвестиционными проектами
3. Управление инновационными проектами
4. Управление IT-проектами
5. Управление организационными проектами

### # Выгода от использования систем управления проектами

1. Сокращение числа проектов, которые не соответствуют стратегии компании
2. Повышение эффективности использования ресурсов
3. Снижение перерасхода бюджета
4. Сокращение процента неудачных проектов
5. Сокращение временных затрат руководителей проекта и др. руководителей

## **26. Виды систем управления проектами.**

### # Виды систем управления проектами

1. Локальные (настольные) - больше для самозанятых подходят

2. Клиент-серверные - требует установки ПО на клиенте.

3. Web-ориентированные

1. Место расположения ПО

1.1. Облачные системы - располагается на сервере разработчика

1.2. На сервере предприятия -

2. По модели ценообразования

2.1. Покупка на весь период использования

2.2. Взымание арендной платы - SAAS

Кроме того, можно выделить также:

- Персональные. Обычно используются для управления домашними проектами. Как правило, это однопользовательские системы с простым интерфейсом.
- Однопользовательские. Однопользовательские системы могут использоваться в качестве персональных или для управления небольшими компаниями.
- Многопользовательские. Предназначены для координации действий нескольких десятков или сотен пользователей. Обычно строятся по технологии клиент-сервер.

## **27. Обзор систем управления проектами. Basecamp, DeskAway, WorkSection, Redmine.**

Критерии для сравнения систем:

- Удобство интерфейса
- Наличие понятного и удобного task-management'a – сервиса постановки задач
- Наличие планировщика
- Удобство средств коммуникаций между пользователями внутри системы
- Наличие инструментов управления бизнес-процессами
- Возможность адаптировать систему под специфическую сферу деятельности
- Наличие инструментов мониторинга и визуализации движения проекта
- Консолидация информации по проекту (создание единого информационного пространства).

Преимущества Basecamp:

- простота и распределенность;
- интуитивный интерфейс;
- интеграция в популярные сервисы для разработки ПО;
- доступна программа для преподавателей и студентов;
- возможность создавать собственные дополнения.

Недостатки продукта:

- строго ограниченный набор услуг;
- трудности в кастомизации: веб-версия слабо подстраивается под ваши вкусы;
- громоздкость: несмотря на простоту в использовании, продукт имеет большое количество возможностей, которые могут оказаться ненужными в небольших проектах.

В DesAway имеется полный список функций по управлению проектами и отсутствие каких-либо намеков на финансовый учет. Однако нет целостности системы: в одном случае страница работает без перезагрузки, в другом похожем ее требует, где-то данные выводятся в модальном окне, а где-то используются всплывающие окна

Интерфейс управления удобен и прост, однако в нем есть несколько спорных моментов. Меню имеет 3-4 уровня, что все-таки многовато. В выпадающем меню пункта задачи можно увидеть пункт все задачи и добавить задачу. Но все же свои задачи интерфейс решает.

Резюме: еще одна система управления проектами с удобным интерфейсом и достаточным функционалом.

В WorkSection всё начинается с задач. Задачи могут иметь подзадачи, к задачам можно прикладывать файлы, оставлять комментарии. Таким образом, всё общение участников проекта происходит через задачи. При этом на странице «Задачи и общение» легко увидеть состояние всех активных задач, и даже с текстом последнего комментария. Иными словами, эта страница всегда показывает как бы текущий срез всего проекта, не больше и не меньше.

Учёт времени в WorkSection привязан к задачам (как и всё остальное). Потраченные часы можно ввести при закрытии подзадачи, а также в любое другое время в разделе «Время». Количество часов для каждой задачи или подзадачи вводится, хранится и отображается одной итоговой цифрой.

Резюме: функциональная и удобная система управления проектами, которая подойдет как небольшим компаниям, так и среднему бизнесу.

Redmine это открытое серверное веб-приложение для управления проектами и отслеживания ошибок. Redmine написан на Ruby и представляет собой приложение на основе широко известного веб-фреймворка Ruby on Rails.

Некоторые возможности Redmine:

- Гибкая система доступа, основанная на ролях;
- Система отслеживания ошибок и создания записей об ошибках на основе полученных писем;
- Ведение новостей проекта, документов и управление файлами, а также создание форумов и вики-страниц проектов;
- Оповещение об изменениях с помощью RSS-потоков и электронной почты;
- Настраиваемые произвольные поля для инцидентов, временных затрат, проектов и пользователей;
- Лёгкая интеграция с системами управления версиями (SVN, CVS, Git, Mercurial, Bazaar и Darcs);
- Поддержка СУБД MySQL, PostgreSQL, SQLite, Oracle.

Недостатки Redmine:

- В Redmine нельзя управлять правами доступа на уровне отдельных полей задачи. Например, на данный момент от клиентов нельзя скрыть оценки времени работы над проектом или информацию о потраченном времени.
- Можно управлять правами доступа на уровне проектов, но нельзя назначить права на какую-то версию проекта или отдельную задачу. Это значит, что если пользователю нужен доступ всего к одной задаче, то придется давать доступ ко всему проекту.
- Если пользователь Redmine получил доступ к проекту, то сейчас нельзя ограничить его активность какими-то отдельными типами задач (трекерами). Например, нельзя разрешить просматривать или создавать задачи только какого-то определенного типа.
- В Redmine не реализовано делегирование задач — нельзя передать задачу другому исполнителю, отметив, что задача должен исполнять он, но оставив себе наблюдение за задачей.

Резюме: система отлично подойдет для компаний, которые хотят хранить данные на своем сервере и имеет возможность доработки (необходимы знания Ruby on Rails).

## 28. Обзор систем управления проектами. Teamwork Project Manager, Intervals, activeCollab.



Team Project Manager очень приятная система управления проектами. Удовлетворяет практически всем требованиям. Отличный интерфейс пользователя, много аяx и именно в там, где это нужно. Система полностью предсказуема, выбираем проект — и четко видим все задачи и общение по нему. Чтобы добавить задачу в список, сообщение или файл в проект, не нужно перезагружать страницу. И так везде.

Имеется русскоязычный интерфейс (переведена большая часть, но иногда встречается и английский). Однако система управления проектами предназначена не только для разработчика, но и для клиента. В этом случае поддержка русского языка — это большой плюс.

Резюме: если вам нужна только система управления проектами — это отличный выбор. Богатый функционал, удобство и поддержка русского языка. Учитывая наличие бесплатной версии на 2 проекта, система подойдет также и для небольших фирм.

Intervals - Интерфейс довольно удобен если с ним разобраться и привыкнуть к нему. Мощные системы фильтров, сортировки и другие возможности управления содержимым позволяют вам гибко управлять вашими проектами, но конечно это сказывается на интерфейсе.

Данная система управления проектами поддерживает все основные функции, но чуть-чуть в большем объеме, чем предыдущие системы. Это наиболее функциональный продукт из рассматриваемых. Кроме того, здесь имеется система выставления счетов на

основе потраченного времени, в которой можно задать даже список ваших специалистов и их ставку.

Ajах здесь тоже используется, но отнюдь не везде где бы хотелось. Так чтобы добавить задачу, вам придется нажать кнопку добавить задачу и дождаться загрузки страницы. С другой стороны при наведении мышки на задачу или проект вы моментально получите информация о нем в окне-подсказке. Так что разработчикам нужно лишь подкорректировать некоторые части интерфейса и получится отличный инструмент для управления проектами.

Резюме: более сложная система управления проектами, чем предыдущие. Делает больше, чем другие, но не всегда лучше и быстрее. В перспективе самый мощный инструмент управления проектами

ActiveCollab - Интерфейс системы строится вокруг проекта. Зайдя в нужный нам проект, мы видим все задачи и файлы с ним связанные. Все очень удобно, красиво и на своих местах. Здесь мы видим и вехи проекта, и задачи/тикеты, и форумы, и файлы, и вики-доски, и управление временем... в общем все, что нужно для управления вашим проектом

Также стоит отметить функционал выставления счетов и контроля их оплат. Но все-таки остается ощущение низкой проработки этого функционала. Но это лучше, чем ничего.

В активе этой системы удобный интерфейс и хорошая функциональность, но главным ее достоинством и отличием от других систем является возможность установки системы к себе на хостинг. Многие отказываются от использования веб-систем управления проектами из-за боязни сохранности личных данных и их приватности. С этой системой вы сможете все свои проекты держать при себе.

Другое достоинство системы вытекает из предыдущего. Раз мы ставим ее к себе на хостинг, значит можем и как угодно модифицировать ее (в рамках лицензии конечно). На сайте уже представлен каталог модулей для activeCollab, а также локализации для него, включая русскоязычную.

Ну и последнее преимущество – имеется панель быстрого добавления задачи или файла в проект. Мелочь, а время экономит прилично.

Резюме: функциональная и удобная система управления проектами с возможностью установки на свой хостинг и поддержкой модулей.

## **29. Технология MVC. Назначение, концепция, основные принципы.**

Model-View-Controller (MVC, «Модель-Представление-Контроллер», «Модель-Вид-Контроллер») — схема разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо

- Модель (Model) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние



- Представление (View) отвечает за отображение данных модели пользователю, реагируя на изменения модели
- Контроллер (Controller) интерпретирует действия пользователя, оповещая модель о необходимости изменений

Важное замечание: концепция MVC не только не привязана к какому-то конкретному языку программирования, она также не привязана и к используемой парадигме программирования. То есть, вы вполне можете проектировать своё приложение по MVC, при этом не применяя ООП.

Отделяет представление от модели, устанавливая между ними протокол взаимодействия подписка/уведомление. Представление должно гарантировать, что внешнее представление отражает состояние модели. При каждом изменении внутренних данных модель уведомляет все зависящие от нее представления в результате чего обновляет себя. Такой подход позволяет присоединить к одной модели несколько представлений, обеспечив тем самым различные представления.

Позволяет также изменять реакцию на действия пользователя. При этом визуальное воплощение может оставаться прежним.

### 30. CMS и CMF. Основные принципы, возможности, отличия, способы реализации в них технологии MVC. Примеры.

Система управления содержимым (Content management system, CMS, система управления контентом) — информационная система или компьютерная программа, используемая для обеспечения и организации совместного процесса создания, редактирования и управления содержимым, иначе — контентом.

CMS обычно состоит из двух основных компонентов:

- приложения для управления контентом (CMA) в качестве внешнего пользовательского интерфейса, позволяющего пользователю добавлять, изменять и удалять контент с веб-сайта без вмешательства веб-мастера,
- приложение доставки контента (CDA), которое компилирует контент и обновляет веб-сайт.

Основные функции CMS:

- предоставление инструментов для создания содержимого, организация совместной работы над содержимым;
- управление содержимым: хранение, контроль версий, соблюдение режима доступа, управление потоком документов;
- публикация содержимого;
- представление информации в виде, удобном для навигации, поиска.

Content Management Framework (CMF) — это каркас (фреймворк управления содержимым) для проектирования систем управления контентом. На их основе создаются системы управления содержимым (CMS), а также веб-приложения.

Если основная задача универсальных систем управления содержимым — простота создания сайта без вмешательства программиста, то есть — конструктор сайтов, то каркас управления содержимым — это конструктор систем управления содержимым (в том числе узкоспециализированных) для программиста. Благодаря такому подходу сайт, созданный с помощью каркаса, по сравнению с сайтом на базе системы управления содержимым, может иметь более простую и безопасную в работе административную панель (в которой отсутствуют функции настройки сайта под любые задачи) и быть менее требовательным к ресурсам системы (каждый модуль реализует именно те функции, которые необходимы в работе сайта).

Большинство современных фреймворков управления содержимым являются реализацией архитектуры [Model-View-Controller](#). Веб-фреймворк обеспечивает бесшовную интеграцию всех трёх слоёв MVC архитектуры. Фреймворки скрывают от программиста детали подключения к базе данных и формирования веб страниц с помощью [шаблонов](#)-представлений, позволяя программисту сконцентрироваться на реализации [бизнес-логики](#)<sup>[2]</sup>.

Процесс создания приложения с использованием MVC-фреймворка заключается в написании классов контроллеров, моделей и представлений, каждый из которых является наследником базовых классов для компонентов каждого слоя.

Многие современные системы управления содержимым (CMS) построены вокруг MVC-паттерна. Такой фреймворк может быть специально написан для системы, примерами могут являться: [Joomla!](#) (начиная с версии 1.5), [Bitrix](#) (начиная с версии 6), [MODX Revolution](#) (начиная с версии 2.0), [SilverStripe](#), [Contao](#), [Frog CMS/Wolf CMS](#) ведётся постепенный перевод на такую архитектуру [TYPO3](#). Другие системы используют фреймворки, популярные сами по себе. Так, написанная на языке [Python](#) система управления содержимым [Plone](#) построена на основе объектно-ориентированного сервера приложений [Zope](#) (и его расширения — CMF<sup>[4]</sup>), коммерческая CMS [ExpressionEngine](#) использует свободный фреймворк [CodeIgniter](#) того же автора.

Такая готовая к использованию система управления контентом, как [Drupal](#), одновременно считается и каркасом для построения таких систем<sup>[6]</sup>, что определяется как возможностью расширения функционала за счёт пользовательских модулей<sup>[7][8][9]</sup>, так и богатством механизмов и абстракций для управления контентом, предоставляемым этой системой

## 31. Маршрутизация(роутинг) и ЧПУ. Принципы реализации и виды маршрутизации.

ЧПУ — веб-адреса, удобные для восприятия человеком.

Достоинства для посетителя очевидны:

- подобные адреса очень легко запомнить;
- можно продиктовать URL по телефону;
- чтобы перейти на уровень вверх достаточно стереть нужную часть пути;
- если человек уже был на вашем сайте и набирает адрес вручную, то он сразу может обратиться к нужному ему документу глядя на URL'ы предыдущих запросов.

Недостатки:

- увеличение затрат ресурсов сервера для большинства реализаций;
- усложнение настройки сайта в связи с необходимостью вмешиваться в конфигурационные файлы веб-сервера.

Оптимальная длина ЧПУ должна составлять от 60 до 80 символов, и, естественно, чем меньше она будет, тем удобнее оперировать таким адресом. Таким образом, не стоит в ЧПУ отображать всю структуру сайта (например, подкатегории товара в интернет-магазине). Достаточно ограничиться первой и последней или одной из подкатегорий.

Хостинг сайта может быть организован на серверах к примеру, Microsoft или UNIX/Linux. данные хостинги будут по разному воспринимать регистр букв в URL. Хостинг на базе Microsoft с легкостью отличит URL [site.ru/OnePage](#) и [site.ru/onepage](#) , а если вас хостинг будет на базе Linux/UNIX, то страница [site.ru/OnePage](#) может быть не найдена из-за двух букв верхнего регистра.

Хорошим методом сокращения длины URL является исключение стоп слов из URL. Поисковые системы не учитывают стоп слова в URL. Это сделает ваш URL более релевантным для поисковых систем.

Две задачи системы маршрутизации – это разбор адресов и их создание.

## 32. Технология AJAX. Основные принципы. Методы библиотеки jQuery, реализующие эту технологию.

**Технология AJAX.** AJAX (аббревиатура от Asynchronous JavaScript and XML) – это **технология** взаимодействия с сервером без перезагрузки страницы. Поскольку не требуется каждый раз обновлять страницу целиком, повышается скорость работы с сайтом и удобство его использования.

При использовании AJAX нет необходимости обновлять каждый раз всю страницу, так как обновляется только ее конкретная часть. Это намного удобнее, так как не приходится долго ждать, и экономичнее, так как не все обладают безлимитным интернетом. Правда в этом случае, разработчику необходимо следить, чтобы пользователь был в курсе того, что происходит на странице. Это можно реализовать с использованием индикаторов загрузки, текстовых сообщений о том, что идет обмен данными с сервером. Необходимо также понимать, что не все браузеры поддерживают AJAX (старые версии браузеров и текстовые браузеры). Плюс Javascript может быть отключен пользователем. Поэтому, не следует злоупотреблять использованием технологии и прибегать к альтернативным методам представления информации на Web-сайте.

Обобщим достоинства AJAX:

- Возможность создания удобного Web-интерфейса
- Активное взаимодействие с пользователем
- Частичная перезагрузка страницы, вместо полной
- Удобство использования

AJAX базируется на двух основных принципах:

- использование технологии динамического обращения к [серверу](#) «на лету», без перезагрузки всей страницы полностью, например с использованием [XMLHttpRequest](#) (основной объект);
  - через динамическое создание дочерних фреймов<sup>[2]</sup>;
  - через динамическое создание тега `<script>`<sup>[3]</sup>;
  - через динамическое создание тега `<img>`, как это реализовано в Google Analytics.
- использование [DHTML](#) для динамического изменения содержания страницы;



Простой пример запроса на нативном JS

```

1  const xhttp = new XMLHttpRequest();
2  const xhttp2 = new XMLHttpRequest();
3  const url = 'https://jsonplaceholder.typicode.com/todos/1';
4  const url2 = 'https://jsonplaceholder.typicode.com/todos';
5  const myFunction = (responseText) => {
6      console.log(responseText);
7  }
8  xhttp.onreadystatechange = (data : Event ) => {
9      if (xhttp.readyState === 4 && xhttp.status === 200) {
10         myFunction(xhttp.responseText);
11     }
12 };
13 xhttp2.onreadystatechange = () => {
14     debugger
15     if (xhttp2.readyState === 4 && (xhttp2.status === 200 || xhttp2.status === 201)) {
16         debugger
17         myFunction(xhttp2.responseText);
18     }
19 }
20 xhttp.open( method: 'GET', url, async: true);
21 xhttp.send();
22 xhttp2.open( method: 'POST', url2, async: true,);
23 xhttp2.setRequestHeader( name: 'Content-type', value: 'application/x-www-form-urlencoded');
24 xhttp2.send(JSON.stringify( value: {name: 'test'}));

```

Этот же запрос но с использованием метода fetch

```

fetch('https://jsonplaceholder.typicode.com/todos/1')
  .then(response => response.json())
  .then(json => console.log(json))

```

```

const url = 'https://example.com/profile';
const data = { username: 'example' };

try {
    const response = await fetch(url, {
        method: 'POST', // или 'PUT'
        body: JSON.stringify(data), // данные могут быть 'строкой' или {объектом}!
        headers: {
            'Content-Type': 'application/json'
        }
    });
    const json = await response.json();
    console.log('Успех:', JSON.stringify(json));
} catch (error) {
    console.error('Ошибка:', error);
}

```

Пример на JQuery

Get

```
26 $.ajax({  
27   url: url,  
28   cache: false,  
29   success: (response) => myFunction(response)  
30 });  
31
```

Post

```
$.ajax({  
  type: 'POST',  
  url: url2,  
  cache: false,  
  data: {name: 'TEST'},  
  success: (response) => myFunction(response)  
});
```

Аналогично для остальных