*Преподаватель: Боклач Богдан Игоревич*

**Моделирование и анализ программного обеспечения**

02.09.22

**Методы организации работы в команде разработчиков. СКВ**

Множество разработок в зависимости от количества участников и типов взаимоотношений между ними может сведено к триаде разработок:

Авторская разработка

Коллективная разработка

Общинная разработка

1. **Авторская разработка**

- принцип создания программных продуктов при котором весь жизненный цикл разработки поддерживается одним единственным человеком.

1. **Коллективная разработка**

Одним из основных вопросов коллективной разработки является разделение труда – от равноправных соисполнителей до организации в виде жёсткой иерархии (например: бригады главного программиста - тимлида).

1. **Общинная разработка**

-Характеризуется тремя основными факторами: децентрализованность разработки; разработка ведётся на базе открытых исходных текстов; большое количество внешних тестов.

**Коллективная разработка**

Бригада равноправных соисполнителей обычно состоит из специалистов, занимающихся примерно подобными задачами в рамках одного проекта. Специализаций в одной бригаде может быть несколько:

1. инженеры разработчики;
2. технические писатели;
3. инженеры тестирования;
4. инженеры качества;
5. специалисты по сопровождению продукта;
6. специалисты по продажам продукта.

**Основные этапы разработки ПП:**

1. Анализ, определение процесса разработки ПП
2. Проектирование – управление проектом разработки;
3. Конструирование – описание целевого программного продукта;
4. Программирование – проектирование продукта;
5. Разработка продукта;
6. Тестирование – тестирование частей ПП;
7. Отладка – интегрирование частей ПП и тестирование продукта в целом;
8. Развёртывание ПП;
9. Выпуск продукта.

**Минимальные функции системы коллективной разработки:**

1. Регистрация изменений, вносимых в проект;
2. Хранения файлов проекта.

**Системы контроля версий.**

СКВ/СУВ – это ПО, предназначенное для работы с постоянно изменяющейся информацией.

**СУВ предоставляет следующие возможности:**

1. Поддержка хранения файлов в репозитории;
2. Поддержка истории версий файлов в репозитории;
3. Нахождение конфликтов при изменении исходного кода и обеспечение синхронизации при работе в многопользовательской среде разработки
4. Отслеживание авторов изменений

**Классификация систем контроля версий:**

1. Централизованные (распределённые) – в централизованных СКВ вся работа производится с центральным репозиторием; в распределённых – у каждого разработчика есть локальная копия репозитория.
2. Блокирующие (неблокирующие) – блокирующие СКВ позволяют наложить запрет на изменения файла, пока один из разработчиков работает над ним; в неблокирующих один файл может одновременно изменяться несколькими разработчиками.
3. Для текстовых данных (бинарных). Для СКВ с текстовыми данными важна поддержка слияния изменений. Для СКВ с бинарными данными важна возможность блокировки.

07.09.22

**Мониторинг работоспособности некоторых из систем контроля версий.**

**Bazaar** (ранее Bazaar-NG) – утилита командной строки bzr, распределённая система управления версиями, разработка которой спонсируется фирмой Canonical Ltd, в последнюю версию по сравнения с предыдущей было внесено более 50 изменений. Данная система разработана в целях облегчения создания и развития проектов для пользователей.

Bazaar – это удобная система контроля версий с приятным интерфейсом, она хорошо подходит для пользователей, которых не привлекает перспектива работы с командной строкой. Имеется множество дополнительных опций и расширений, что позволяет настроить программу под свои нужды.

**Mercurial** (англ. «Подвижный») – распределённая система управления версиями, способная функционировать на многих операционных системах и различных аппаратных платформах, разработанная для эффективной работы с очень большими кодами.

Говоря о Mercurial, следует отметить, что простой и отточенный интерфейс, набор команд, возможность импорта репозиториев с других систем контроля версий сделают переход на данную программу безболезненным и быстрым, а её надёжность и скорость работы позволяют пользоваться им для контроля версий огромных проектов. Всё это позволяет Mercurial стать достойным конкурентом git’а.

**Git** – распределённая система управления версиями файлов. Код программы был написан на языке “C”, проект создан Линусом Торвальдсом в 2005 году для управления разработкой ядра Linux, является общедоступным программным обеспечением. Данная система была введена многими ведущими разработчиками, используется в известных Linux-сообществу проектах.

В свою очередь Git – это гибкая, удобная и мощная система контроля версий, способная удовлетворять абсолютное большинство пользователей. Git – один из лидеров систем контроля версий.

**Concurrent Versions System** (или CVS, англ. “Система Одновременных Версий”) – представляет собой программный продукт, который относится к разряду систем управления версиями. Программа хранит историю изменений исходного кода программного обеспечения, тем самым облегчая совместную работу программистов над одним проектом. CVS популярна в мире открытого программного обеспечения.

Несмотря на то, что CSV достаточно устарела и обладает весомыми недостатками, она всё ещё является одной из самых популярных систем контроля версий и отлично подходит для управления малыми проектами, не требующих создания нескольких параллельных версий, которые надо периодически соединять.

09.09.2022

**Создание и изучение возможностей репозитория проекта (ЛР 1)**

Под **системой контроля версий** понимается механизм сохранения промежуточных состояний кода, разрабатываемого программного обеспечения. То есть с помощью этой системы программист может управлять своими файлами во времени: смотреть историю изменения файлов и каталогов, возвращаться к более ранним версиям кода, объединять несколько версий файлов.

Основной **областью применения контроля версий** является коллективная разработка (чаще всего программы, область применения программирования которых не ограничивается). Однако, и для разработчика одиночки контроль версий может быть полезен.

Для решения этих проблем удобно использовать системы контроля версий, например SubVersion (SVN) – одна из наиболее известных систем версионного управления исходным кодом, применяется во многих OpenSource проектах. Подобные системы позволяют узнать кто, когда и как именно вносил изменения в файлы проекта. Всё это значительно облегчает групповую разработку ПО.

Система контроля версий имеет две части: сервер и клиент. Каждый клиент имеет свою локальную копию исходных кодов, с которой работает разработчик (разработчики работают только с ними).

**Минусы локальных копий:**

* Объём;
* Одновременное редактирование одного файла.

**Модели системы контроля версий:**

* Блокировка – изменение – разблокировка  
  (когда кто-то начинает работу с файлом, этот файл блокируется, и другие пользователи теряют возможность его редактирования.  
  Недостаток – файлы могут оказаться надолго заблокированными, что приводит к простоям разработки проекта);
* Копирование – изменение – слияние  
  (каждый разработчик свободно редактирует свою локальную копию файла, после чего выполняется слияние и изменения.  
  Недостаток – может возникнуть необходимость разрешения конфликтов между изменениями файлов);

Работа с рабочей копией файла:

При работе с рабочей копией часто необходимо понять, какие файлы разработчик заменил, добавил, удалил, переименовал, а также изменения файлов другими разработчиками. Теперь после извлечение рабочей копии из хранилища в Subversion можно заметить, что значки в проводнике Windows немного изменились, и это одна из причин популярности TortoiseSVN. Она добавляет «пометку» на значке для каждого файла, которая накладывается на исходный значок файла.

16.09.2022

**Цели, корректность и направления анализа Программных Продуктов**

**Общая характеристика тестирования и его цикл**

**Тестирование ПО** – процесс исследования, испытания программного продукта, имеющий своей целью проверку соответствия между реальным поведением и её ожидаемым поведением на конкретном наборе тестов, выбранным определённым образом (ISO/IECTR 19757 2005).

Тестирование представляет собой деятельность по проверке программного кода и документации. Она должна заранее планироваться и систематически проводиться специально назначенным независимым тестировщиком. Работа тестировщика начинается до утверждения спецификаций требований. Он проверяет требования к ПП на полноту и возможность тестирования, определяет методы тестирования.

Одновременно с началом этапа планирования и создания спецификаций требований тестировщик разрабатывает стратегию тестирования. После утверждения спецификаций требований им разрабатывается и детализируется план тестирования. Тогда же тестировщик создаёт наборы тестов для проведения интеграционного и системного тестирований. Тестирование завершается созданием отчёта о тестировании, в котором представляются все результаты его проведения.

Для каждого программного изделия должен существовать набор тестов, проверяющий его корректность. Существует несколько уровней тестирования, позволяющих полностью проверять программное изделие.

Создание базовой версии ПП

Проведение тестирования

Анализ результатов

Создание отчёта

Выявлены ли ошибки?

Исправление ошибок

нет

да

**Уровни тестирования:**

1. Модульное (осуществляет сам разработчик на этапе разработки);
2. Интеграционное (отвечает независимый тестировщик);
3. Системное (отвечает независимый тестировщик);
4. Выходное (отвечает независимый тестировщик);
5. Приёмочное (выполняется совместно с представителями заказчика).

Тестирование 1-4 уровней проводится внутри организации.

**Цикл тестирования** – совокупность действий, выполняемых тестировщиком с момента передачи базовой версии ПП тестировщику для интеграционного, системного и приёмочного тестирования до момента успешного завершения тестирования.

На каждом проходе цикла тестирования:

* Создаётся базовая версия ПП, подлежащего тестированию;
* Создаётся отчёт о ходе тестирования;
* Метрики тестирования (заносится в базу данных проекта).

**Виды тестирования**

1. **Модульное тестирование** – процесс проверки отдельных программных процедур и подпрограмм, входящих в состав программ или программных систем.

Модульное тестирование производится непосредственным разработчиком и позволяет проверять все внутренние структуры и потоки данных в каждом модуле. Этот вид тестирования является частью этапа разработки. При модульном тестировании выполняется набор тестов, определяемый разработчиком так, чтобы охват тестированием каждого модуля был не менее 70-75%.

**Элементы модульного тестирования:**

* Синтаксическая проверка – проверка с использованием некоторого инструментального средства для выявления синтаксических ошибок в программном коде;
* Проверка соответствия стандартам кодирования – проверка кода на соответствие стандартам кодирования компании;
* Технический обзор программного кода.

После успешного завершения модульного тестирования все изменённые модули и наборы тестов сохраняются в БД проекта.

1. **Интеграционное тестирование** – проводится для проверки совместной работы отдельных модулей и предшествует тестированию всей системы как единого целого.

В ходе интеграционного тестирования проверяются связи между модулями, их совместимость и функциональность. Оно осуществляется независимым тестировщиком и входит в состав этапа тестирования.

**Элементы интегрированного тестирования:**

* Проверка функциональности – проверка соответствия отдельных функций, выполняемых совокупностями модулей, функциям, заданных в спецификациях требований;
* Проверка промежуточных результатов – проверка всех промежуточных результатов и файлов на наличие и корректность;
* Проверка интеграции – проверка того, что модули передают друг другу информацию корректно.

Ошибки, выявленные в ходе интеграционного тестирования, заносится в БД ошибок.

Результаты интеграционного тестирования включаются в отчёт о ходе тестирования при завершении цикла тестирования.

1. **Системное тестирование** – предназначен для проверки программной системы в целом её организации и функционирования на соответствие спецификациям требований заказчика. Его проводит независимый тестировщик после успешного завершения интеграционного тестирования.

**Элементы системного тестирования:**

* Граничное тестирование – тестирование в граничных условиях;
* Прогоночное тестирование – тестирование всех функциональных характеристик реальной работы системы;
* Целевое тестирование – тестирование на целевой платформе (по возможности);
* Проверка документации – проверка пользовательской документации на корректность;
* Другие тесты, определяемые тестировщиком.

Ошибки, выявленные при системной тестировании, заносятся в БД проекта. Результаты системного тестирования включаются в отчёт о ходе тестирования.

21.09.2022

1. **Выходное тестирование** –завершающий этап тестирования, на котором проверяется готовность ПП к поставке заказчику. Данный вид тестирования проводит независимый тестировщик.

**Элементы выходного тестирования:**

* Проверка инсталляции – проверка на ясность и корректность инструкций по инсталляции;
* Проверка документации – проверка того, что вся необходимая документация полностью подготовлена и готова к передаче заказчику.

1. **Приёмочное тестирование** – проводится организацией, отвечающей за инсталляцию сопровождение программной системы и обучение конечного пользователя.

**Программная ошибка** – расхождение между программой и её спецификацией, причём только тогда, когда спецификация существует и верна.

**Программная ошибка** – ситуация, когда программа не делает того, что пользователь от неё обоснованной ожидает.

**Категории программных ошибок:**

1. **Функциональный недостатки** – программа не делает того, что должна/делает это некорректно. Возможно только при наличии спецификации;
2. **Недостатки пользовательского интерфейса** – удобство и правильность работы пользовательского интерфейса;
3. **Недостаточная производительность** – скорость работы, несоответствующая спецификации или ощущениям заказчика;
4. **Некорректная обработка ошибок** – отсутствие информирование программой об обнаружении ошибки;
5. **Некорректная обработка граничных условий** – неправильное поведение программы на границах диапазонов параметров программы;
6. **Ошибка вычислений** – ошибки, вызванные использованием неправильных формул, вычислений, ошибки округления и т.д.;
7. **Ошибки управления потоком** – нарушение порядка выполнения действий.
8. **Ситуация гонок** – событие выполняется первым, при том, что оно должно всегда выполняться вторым;
9. **Перегрузки** – несоответствие реальных возможностей программы и требований к спецификации её ресурсов, поведение программы в ситуациях несоответствия;
10. **Некорректная работа с аппаратурой компьютера** – неправильная работа с данными аппаратуры, отправка неправильных данных аппаратуре.

**Параметры документации:**

* Точность;
* Полнота;
* Ясность;
* Простота использования;
* Соответствие ПП.

30.09.2022

**Reverse Engineering  
Обратная разработка и взлом ПО**

**Задачи обратной разработки:**

* Установить логику программы с закрытым исходным кодом;
* Воссоздать программу, аналогичную проприетарной;
* Избавить проприетарную программу от ненужного функционала (проверка производителя/лицензии).

**Программа:**

* Исполняемый файл популярных ОС и архитектур;
* Байт-код виртуальной машины (Java/.NET);
* Интерпретируемый код (PHP/Python/Perl);

**Исполняемый файл** – набор инструкций процессора вместе с данными, необходимыми для работы программы.

**Форматы исполняемых файлов:**

* PE (Portable Executable) – Windows (расширение файла: .exe);
* ELF (Executable and Linkable Format) – Linux;

Разделяемые библиотеки .dll/.so имеют схожий формат, хотя их обратная разработка затруднена непростой отладкой.

**Запуск исполняемого файла:**

* Windows – двойной щелчок/по имени из командной строки;
* Linux – например «chmod +x file» («./file» вернёт путь, а не файл).

Исполняемый файл состоит из сегментов, секций и т.д. Они разделяют данные, код, константы и пр. В Linux это можно посмотреть командой «readelf». IDA красит различные секции в соответствующие цвета.

**IDA:**

* 1200$;
* Умеет дизассемблировать почти всё;
* Стандарт индустрии;
* Декомпилятор HexRays.

**HexRays:**

* Дороже;
* Умеет декомпилировать.

**Отладчик:**

* Позволяет выполнять программу пошагово, смотреть регистры, инструкции на языке ассемблера;
* Для Windows – OllyDdg и x64dbg;
* GNU Debugger (gdb);
* IDA.

**Z3:**

* Позволяет обращать хеш-функции, LFSR, решать уравнения;
* Есть различные решатели систем условий.

06.10.2022

**Ванины вирусы:**

**Code Red** – червь, изменяет содержание веб-страниц и распространяется через вирусы. Выполнял DDoS атаки. Самый известный прецедент – атака сервера белого дома.

**LoveLetter** – пользователи получали «любовное письмо», а при открытии получали червя, который рассылал себя всем контактам. Записывал графические файлы и похищал пароли.

**Sobig.F** – троян и самокопирующийся червь. Нагружал системы и очень быстро копировался.

**MyDoom** – червь. Рассылался через ошибку об отправки сообщения, а затем рассылал себя всем контактам. Замедлял скорость интернета.

**StuxNet** – червь, использует уязвимости в Windows.

**Winlock** – вредоносная программа, блокирующие или затрудняющие работу с ОС, требуют перевод денег за разблокировку.

**Принцип работы** – после запуска прописывается в autorun, блокирует сочетания клавиш, explorer подменяется вирусным файлом в реестре.

**Дизассемблеры**

IDA (freeware/pro), HexRays Decompiler.

**Методы анализа ПО без использования исходного кода**

**Особенности современного ПО:**

* Сложная модульная система;
* Технологическое разнообразие:
  + Native и Managed модули;
  + Mixed code.

**Внедрение Managed кода в современные ОС:**

* Microsoft Windows:
  + .NET Framework;
  + Java;
* Linux:
  + Java;
  + Mono;
* Мобильные платформы:
  + Java;
  + .NET Framework;
  + Mono.

**Важность анализа ПО без исходного кода:**

* Доступ к исходным кодам часто ограничен;
* Влияние настроек и особенностей компилятора на фактический исполняемый на процессоре код.

**Средства реверс-инжиниринга:**

* HexRays, IDA Pro (Managed и Native);
* Red Gate .NET Reflector (Managed);
* IIdasm – (Native);
* Другие.

07.10.2022

**Методы серого ящика**

**Дизассемблеры:**

* HexRays, IDA Pro (Managed и Native);
* dumpbin – (Microsoft Windows SDK; Native);
* IIdasm – (Native);
* Другие.

**Декомпиляторы:**

* Red Gate .NET Reflector – Managed код;
* JetBrains dotPeek – Managed код;
* Другие.

Полная декомпиляция Native кода в высокоуровневый программный код на данный момент не реализована. Имеются лишь механизмы частичного восстановления структуры алгоритма.

**Отладчики:**

* WinDBG – Managed и Native;
* OllyDBG – Native;
* Другие.

**Средства мониторинга:**

* SusinternalsSuite:
  + Procexp;
  + Procmon;
  + Portmon;
  + Другие.

**Сравнение:**

* Дизассемблированные листинги в общем случае менее удобны для анализа;
* Дизассемблированные листинги дают большую полноту понимания процесса выполнения ПО.

**Современное ПО:**

* Native модули;
* Managed модули;
* Mixed код.

**Некоторые особенности Managed кода:**

* Генерируемый JIT-компилятором код также является динамически создаваемым объектом и располагается в куче;
* В режиме Run-Time код и данные могут размещаться по адресам, выбранным JIT компилятором;
* Код CIL функции можно обнаружить по базовому адресу загрузки модуля + смещение функции.

**Заключение:**

Современное ПО:

* Native код;
* Managed код;
* Mixed код (компилятор Microsoft Visual C++):
  + #include <mscorlib.dll>;
  + #pragma managed;
  + #pragma unmanaged;
* JIT компилятор выбирает адреса размещения компилированных функций самостоятельно;
* Управляемая куча имеет атрибуты RWX;
* IL код модуля можно найти по базовому адресу загрузки + смещение функции.
* Различия в интерпретации результатов работы дизассемблеров и декомпилятора;
* Смешивание Native и Managed модулей в рамках даже одного проекта;
* Mixed код в некоторых модулях;
* Важно развитие средств анализа программного обеспечения, способных обрабатывать не только Native и Managed, но и Mixed сборки. Результаты интерпретации должны иметь гомогенная структура.

12.10.2022

**Система контроля версий**

**Библиотеки классов** также являются важной частью современных систем программирования, базирующихся на ООЯП.

**Недостаток библиотеки классов** – все её классы должны быть написаны на том же ЯП, на котором пишется программа, куда интегрируются библиотечные классы.

**В библиотеке классов различают:**

* Конкретные классы;
* Абстрактные классы, иерархии классов;
* Шаблоны классов, иерархии шаблонов классов.

Библиотеки классов включаются в программу на этапе компиляции и компилируются со всей программой вместе.

**Библиотеки компонентов** – это библиотеки готовых откомпилированных программных модулей, предназначенных для использования в качестве составных частей программ, и которыми можно манипулировать во время разработки программ.

**Компоненты**:

* Локальные – находящиеся на той же ЭВМ, где создаётся ПП);
* Распределённые – расположенные на сервере и доступные по сети ЭВМ).

**Примерных технологий, использующих библиотеки компонентов:**

* Технология CORBA (Common Object Request Architecture) от международной группы OMG позволяет использовать программные компоненты, размещённые как локально, так и дистанционно. Использование CORBA-компонент не зависит от языка, на котором они были написаны;
* Технология COM (Common Object Model) от компании Microsoft под ОС Windows позволяет использовать локально размещённые компоненты, независимо от языка их реализации. Её развитие привело к распределённой архитектуре DCOM (Distributed COM), а затем к ActiveX;
* Технология Java Beans от Sun Microsystems позволяет использовать компоненты, написанные на языке Java. Так как реализация Java-машины существует почти для всех ОС, отсутствует жёсткая привязка к конкретной ОС.

Динамические Библиотеки подключаются к программе в ходе её выполнения.

На этапе компоновки программы редактор связей, встречая вызовы функций ДБ, вместо процедуры связывания формирует таблицу точек вызова функций ДБ для последующей операции динамического связывания. Таким образом, процесс полной компоновки завершается уже на этапе выполнения целевой программы.

**Преимущества ДБ:**

* Не требуется включать в программу объектный код часто используемых функций, что существенно сокращает объём кода;
* Различные программы, выполняемые в некоторой ОС, могут пользоваться кодом одной и той же библиотеки, содержащейся в ОС;
* Изменения и улучшения функций библиотек сводится к обновлению только содержимого ДБ, а уже существующие тексты программ не требуют перекомпиляции (этот же факт может оказаться недостатком, если при модификации функций меняется логика их работы, поэтому использование ДБ накладывает определённые обязательства как на разработчика программы, так и на создателя библиотеки.

Как правило, динамически подключаются системные функции ОС и общедоступные функции программного интерфейса (API).

Существует возможность создавать свои ДБ для использования при разработке прикладных программ.

**Стандартная библиотека должна:**

* Обеспечивать поддержку свойств языка (например, управление памятью, предоставление информации об объектах во время выполнения программ);
* Предоставлять информацию о зависящих от реализации аспектах языка (например, о максимальных размерах целых значений);
* Предоставлять функции, которые не могут быть написаны оптимально для всех вычислительных систем на данном языке программирования (например, sqrt() или memmove() пересылка блоков памяти);
* Предоставлять программисту нетривиальные средства, на которые он может рассчитывать, заботясь о переносимости программ (например, средства работы со списками, функциями сортировки, потоки ввода/вывода);
* Предоставлять основу для расширения собственных возможностей, в частности, соглашения и средства поддержки, позволяющие обеспечить операции для данных, имеющих определяемые пользователями типы, в том же стиле, в котором обеспечиваются операции для встроенных типов (например, ввод/вывод);
* Служить основой и теоретическим базисов других библиотек.

**Компоненты стандартной библиотеки должны:**

* Иметь общезначимый характер (структуры данных и алгоритмы для работы с ними – стек, очередь, список, …, сортировка, поиск, копирование, …), быть важными и удобными для использование всеми программистами;
* Быть настолько эффективными, чтобы у пользователей библиотеки не возникало потребности заново программировать библиотечные средства;
* Быть независимыми от конкретных алгоритмов или предоставлять возможность указывать алгоритм в качестве параметра;
* Оставаться элементарными, чтобы не терять эффективности из-за излишних усложнений или попыток совместить различные функции в одной;
* Быть безопасными (устойчивыми к неправильному использованию, использование библиотеки не должно провоцировать ошибки, а наоборот, снижать их вероятность);
* Обладать достаточной полнотой (завершённостью), чтобы ни у кого не возникало желания что-то заменить или доопределить;
* Обладать удобной и безопасной системой умолчаний;
* Поддерживать общепринятые стили программирования;
* Обладать способностью к расширению, чтобы работать с типами, определяемыми пользователем, было так же хорошо, как и со встроенными (базовыми) типами (сочетаемость с базовыми типами данных и базовыми операциями).

08.11.2022

**Введение в программирование на Java**

**JDK (Java Development Kit).** Основные инструменты, необходимые для создания программ на языке Java, включены в JDK – это комплект разработчика приложений на языке Java.

**JDK** – программное обеспечение с открытым исходным кодом, его можно скачать свободно и бесплатно.

**Состав JDK:**

* Компилятор Java (javac);
* Документация;
* Примеры;
* Утилиты;
* Исполнительная система Java (JRE Java Runtime Environment).

**Состав JRE:**

* Виртуальная Java-машина (Java-интерпретатор);
* Библиотеки классов Java.

**Редакции JDK:**

* Standart Edition (SE) – для создания и исполнения приложений для индивидуального использования (или использования в масштабах малого предприятия);
* Enterprise Edition (EE) – для создания коммерческих приложений для крупных и средних предприятий;
* Micro Edition (ME) – для разработки приложений для устройств, ограниченных в ресурсах (телефонов, кпк и т.д.)

**IDE для Java:**

* NetBeans IDE;
* Sun Java Studio Creator;
* IntellyJ IDEA;
* Borland JBuilder;
* Eclipse.

**Состав IDE:**

* Специализированный текстовый редактор, который облегчает форматирование текста программы, подсвечивает синтаксис и предоставляет прочие удобства;
* Отладчик – программный комплекс для поиска и исправления ошибок в программе;
* Фоновый компилятор – указывает на синтаксические ошибки ещё в процессе набора текста программы;
* Справочная система.

Для использования программы на компьютере должна быть установлена JRE, Java-машина интерпретирует байтовый код, учитывая особенности данной ОС (исполняемый код НЕ создаётся).

Машинный код  
\*.class

Виртуальная Java машина

Компилятор

Исходный код  
\*.java

Текстовый редактор

НО: Исполняемый файл (нативный бинарный код)…

**Java как формальный язык:**

* Алфавит – конечное упорядочивание множество символов;
* Синтаксис – правила записи конструкций (операторов) из символов алфавита;
* Семантика – смысловая сторона языковых конструкций;
* Прагматика – последствия практического применения языка.

**Алфавит языка Java:**

* Зарезервированные слова;
* Символы для записи идентификаторов:
  + Буквы;
  + Десятичные цифры;
  + Символ;
* Разделители;
* Специальные символы.

**Имя (идентификатор)** – последовательность букв, цифр и символа «\_» (нижнее подчёркивание).

Поименован должен быть каждый объект программы для того, чтобы к нему можно было обратиться.

Объявление объекта должно предшествовать его вызову.

**Структура Java-программы:**

* Программа – это класс (Java – это объектный процесс);
* Класс – совокупность полей и методов;
* Поля – данные, переменные;
* Методы – реализуют алгоритмы обработки данных (подпрограммы, процедуры, функции).

**Структура Java программы.**

Компилятор Java чувствителен к регистру.

Переменная ячейка памяти для хранения данных.