netstat -a посмотреть все открытые порты на компе

проблемное место – ставим туда курсор и нажимаем alt+enter и СР сама предложит нам выход из проблемы

механизм продолжения ctrl+space

Выравнивание кода – menu-code-reformat code “Ctrl-Alt-L”

Отступы для форматирования кода можно настраивать – урок 8 модуль 1

Можно отключить предложения среды разработки добавлять файлы в репозиторий. урок 8 модуль 1

Два основных правила – классы всегда с большой буквы все остальное(методы переменные и так далее) с маленькой. И принято для создания названий использовать так называемый кэмелкэйс.

Когда связка слов идет через большую букву – такПринятоВДжаве – а\_так\_в\_джаве\_не\_принято

В джаве реализован следующий принцип – если действия совершаются с какими то типами данных, например с интегер, то джава считает что и результат должен получаться в интегер. То есть попытка вывести результат деления 1 на 2 выдаст на выходе 0. То есть округление 0.5 до ближайшего целого.

И в таких ситуациях надо явно указать в чем проводить вычисления

Еще если идет сложение двух разных типов то джава автоматически приводит к тому типу к котору можно привести

Операция «2»+2 выдаст на выходе 22 и это будет строка

То есть при контакенации(сращивании) двух строк если в качестве одного из аргументов контаккенации выступает тип отличный от строки то он приводится к строчному значению.

Если мы запишем что то типа «2+2=»+2+2 то на выходе мы получим 2+2=22

Так как джава начнет вычисление со среднего знака + и первую двойку сразу приведет к строке

Чтобы получить правильный ответ надо этот пример записать как «2+2=»+(2+2)

Есть понятие значение (value) и есть понятие выражения. Значение это то что упростить нельзя. Это нейкое фиксированное состояние чего то. При этом есть литеральные значения, то есть те которые прописаны в программе явным образом и они известны на момент написания программы. И вычисляемые значения.

Есть переменные variable. В программировании переменная это идентификатор, указывающая на какое то конкретное значение. В отличии от математики где с переменными можно совершать какие то действия упрощая выражения без использования конкретных значений, в программировании программа не может оперировать переменными без значений. И всегда переменная имеет какое то значение. Фактически конкретная переменная является синонимом какого то значения хранящегося в памяти.

Для того чтобы разобраться с типами переменных надо разобраться с типами значений.

Есть целые числа, с плавающей точкой, строки,

Джава относится к языкам со строгой типизацией, то есть соответствие типов переменных и их значений вычисляется и проверяется еще на этапе компиляции.

Дебаггин!!! Классная штука

Функция может ничего не возвращать и не принимать

Тогда запись будет такая

Модификаторы воид название()

Если она принимает какие то параметры но ничего не возвращает тогда запись такая

Модификаторы воид название(типПараметра переменна, …)

Если функция что то принимает и еще и возвращает то так

Модификаторы типВозвращаемогоПараметр «переменная» название(типПараметра «переменна», …)

Класс это описание структуры однотипных объектов.

Для построения теста мы используем тестнг, при этом можно в тесте использовать просто слово ассерт и дальше подать какое то булевское значение. Или вызвать метод в классе Assert например assertEquals(object, object) в данном случае ассерт проверит не только значения и их равенство но и их типы.

После добавления в проект файла с записанного рекордера весь этот новый класс горит красным. И начать надо с подключения библиотек селениума.

Для этого отправляемся в центральный репозиторий мавена search.maven.org и в строке поиска вводим селениум

Для превращения части кода в функцию надо использовать рефакторин

Для параметризации в коде метода там где передается конкретный параметр выделяем этот параметр, вызываем рефакторинг-экстракт-параметр, даем ему имя и он появляется в тексте вызова метода.

Если у нас какой то метод принимает слишком много параметров то гораздо проще и экономнее передать туда объект с параметрами. Типа юзер например, или контакт. Для этого рефакторинг-экстракт-параметр обджект.

Код-оптимайз импорт = удаление серых строчек импорта

Делегирование – если все вспомогательные методы хранить в тестбэйз то этот класс со временем станет очень большим и не читаемым. Поэтому делаем так – создаем над тестбэйсом родителя под названием аппликейшен менеджер и поднимаем в него все вспомогательные методы. В тестбэйс оставляем только бифотест и афтетест. Затем делаем делегирование. Для этого рефактор – реплэйс инхеританс УИС делегейшн. При этом будет создано новое поле(атрибут) класса тестбэйс, скажем апп, галочки ставить не нужно. И везде будет идти обращение уже не к методам напрямую а к объекту апп и через него к методам.

При попытке перенести из класса аппликейшн менеджер часть методов в групхелпер, при втором рефакторинге(делегейт) выскочит ошибка, надо в диалоговом окне с перечнем методов поставить галочку - - generate getter for delegated components.

При таком варианте рефакторинга ссылка на драйвер переедет в группхелпер. А ее надо вернуть обратно в аппликейшн менеджер. Средствами рефакторинга сделать это нельзя. Поэтому создаем в аппменеджер атрибут иебдрайвер. В методе инициализации в апп менеджер в ручную убираем указание на класс группхелпер. И после перехода на страницу приложения инициализируем создание объекта класса групхелпер. И в качестве параметра при создании объекта групп хелпер передаем объект вебдрайвер. Так как у нас нет такого конструктора в группхелпер, то нам среда разработки предложит его создать. Соглашаемся и записываем в коструторе зис.вебдрайвер=вебдрайвер. Кроме этого в аппликейшнменеджере в строке объявления групп хелпер убираем слово файнал.

Так как в данном случае мы разделяем логику приложения на два слоя – логика тестов и логика приложения, то есть смысл код разделить физически – раскидав по разным пакетам.

Выделяем нужный кусок кода правая кнопка-рефакторинг-экстракт-метод

Переформатирование кода – код-реформат