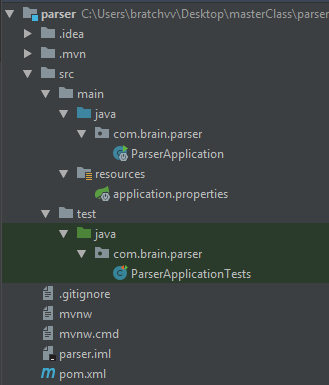
test github repo: <https://github.com/Bratchvv/brainLearnParser>

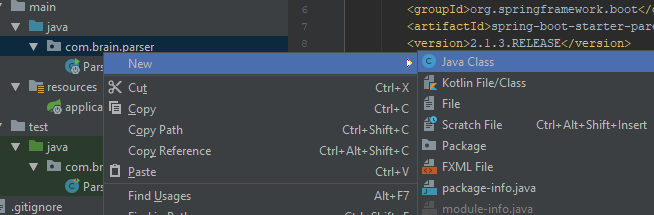
1. Гуглим «spring boot generator».
2. Открываем <https://start.spring.io/>
3. Оставляем все по умолчанию.
   1. По желанию, можно поменять
      1. *group -> ”com.brain”*
      2. *parser -> “parser”*
4. Нажимаем кнопку «Generate Project»
5. Выбираем папку, куда сохранить архив с каркасом проекта
6. Распаковываем содержимое архива.
7. Открываем проект с помощью Intelijj IDEA (file -> open -> {папка с проектом} -> pom.xml)
8. Выбираем вариант «Open as project»
9. Проверяем работу сгенерированного проекта.
   1. Проверяем структуру:

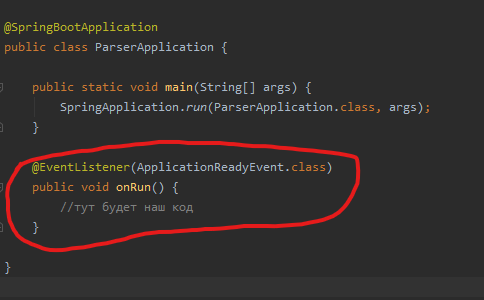


* 1. Находим enter-class : *src/main/java/com/brain/parser/ParserApplication.java*
  2. Запускаем приложение. ПКМ на классе -> Run ’ParserApplication’
  3. *Проверяем консоль на отсутствие ошибок*

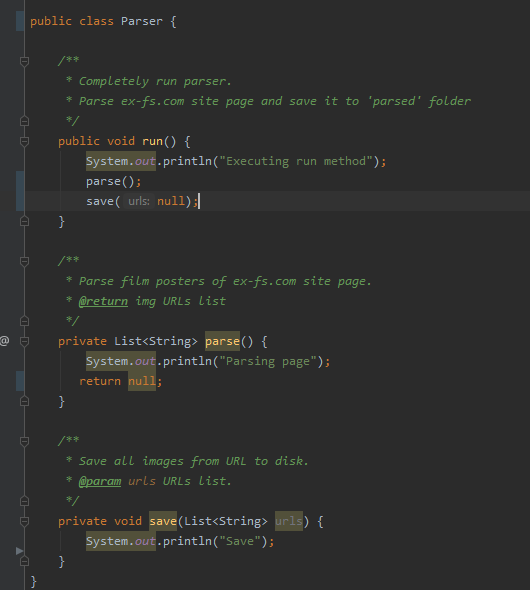
1. Подключаем библиотеку JSOUP
   1. Находим JSOUP в maven repository. Гуглим «jsoup maven»
   2. Открываем <https://mvnrepository.com/artifact/org.jsoup/jsoup/1.9.1>
   3. Копируем dependency
   4. Открываем pom.xml в корне нашего проекта
   5. Добавляем скопированную dependency в блок dependencies, рядом с другими.
   6. Появится Push уведомление “Maven projects need to be imported”. Выбираем «Import Changes»



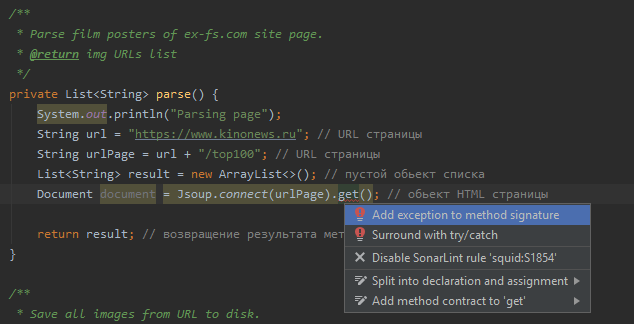
1. Теперь у нас есть все необходимое для создания парсера сайтов. Выбираем сайт, например, <https://www.kinonews.ru/top100>
2. Создаем класс для парсинга сайта.
   1. ПКМ на имени пакета в котором хотим создать класс:   
      
   2. Указываем параметры класса. Имя Parser, тип оставляем Class
   3. Проверяем наличие нового класса в пакете.
3. Заполняем класс «заглушками методов»
   1. создаем будущий метод, для запуска парсера:  
        
      public void run() {  
       // todo  
      }
   2. создаем будущий метод, для сохранения распарсенных картинок сайта  
        
       private void save() {  
       // todo  
      }
   3. создаем будущий метод парсинга html страницы, для получения нужных катинок  
        
      private List<String> parse() {  
       // todo  
       return null;  
      }  
        
      после написания этого метода, компилятор java попросит импортировать сторонний класс List. Устанавливаем курсор мыши на слово List (на середину), нажимаем сочитание клавиш alt + Enter.   
      После этого в классе появится новый импорт: import java.util.List;
4. Вызываем заглушки методов
   1. Открываем класс ParserApplication
   2. Создаем метод для запуска нашей бизнес-логики

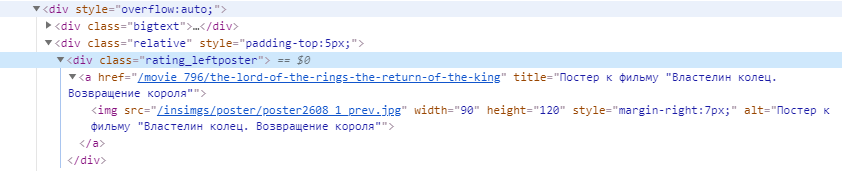


* 1. Дописываем в тело этого метода создание объекта нашего парсера  
     ExFsParser parser = new ExFsParser();
  2. Вызываем метод запуска парсера:  
     parser.run();
  3. Пробуем запускать, аналогично п. 9.с. Вывод должен быть аналогичным.

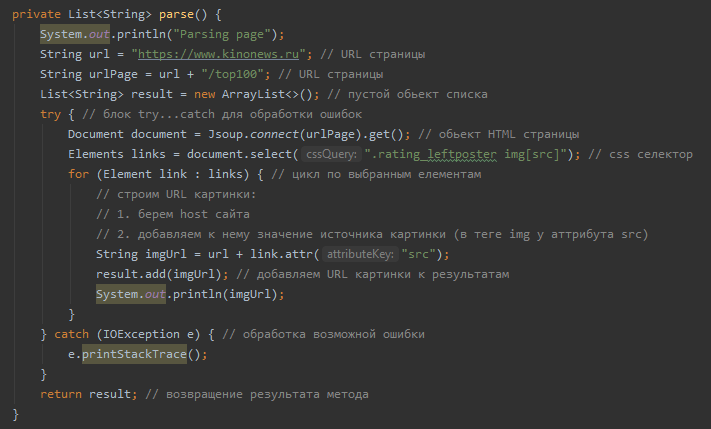
1. Добавляем в методы класса ExFsParser тестовые сообщения для имитации работы
   1. В методе run, вместо комментария «//todo», пишем:   
      System.out.println("Executing run method");  
      Примечание: в Intelijj IDEA существует множество полезных фун-ций, одна из них это сокращения часто используемых команд, например sout для нашего случая.
   2. В методе save, вместо комментария «//todo», пишем:   
      System.out.println("Try to save");
   3. В методе parse, вместо комментария «//todo», пишем:   
      System.out.println("Parsing page");
   4. Пробуем запускать. В логе можно найти строчку "Executing run method".  
      Остальных нет, потому что мы нигде не вызывали их методы.
   5. В методе run вызываем метод parse() и save()
   6. Пробуем запустить еще раз. Теперь на консоле выводятся все нужные сообщения.  
      Наше приложение обрело свой функциональный каркас.
2. Реализуем основной метод для парсинга. Метод parse();  
   1. Создаем строку с URL нужной страницы.  
      String url = "[https://www.kinonews.ru](https://www.kinonews.ru/)";  
      String urlPage = url + "/top100";
   2. Создаем переменную для хранения распарсеных ссылок на картинки:  
       List<String> result = new ArrayList<>();  
        
      Примечание класс ArrayList еще не импортирован, его нужно импортировать, аналогично п. 13.с.
   3. Выполняем чтение HTML страницы с помощью библиотеки JSOUP:  
      Document document = Jsoup.connect(url).get();  
        
      Примечание: классы Document и Jsoup необходимо импортировать, аналогично предыдущему пункту.   
      Импорты:  
      import org.jsoup.Jsoup;

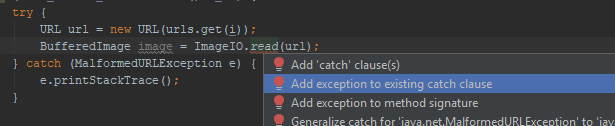
import org.jsoup.nodes.Document;

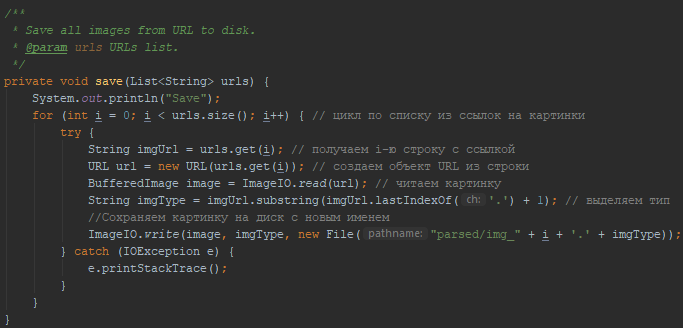
* 1. Обращаем внимание что IDE подчеркнула ошибкой метод get();  
     Intelijj IDEA также часто дает подсказки и предлагает автоматическое решение некоторых проблем.   
     Ставим курсор на этот метод и нажимаем alt+Enter.  
       
     Выбираем пункт “Surround with try/catch”.   
     Проверяем наличие изменений.   
     Блок try…catch нужен для обработки возможных ошибок.
  2. Итак, получение HTML документа у нас работает. Пора написать нужный селектор:  
     Elements links = document.select(".custom-poster img[src]");  
       
     Примечание: Elements нужно импортировать в класс, уже известным способом.
  3. Строка ".rating\_leftposter img[src]" – это css селектор.  
     Открыв «исходники» html страницы (в Chrome -> ctrl + U), попробуем найти (ctrl + F)  
     элементы с таким css классом. Ищем текст «custom-poster».  
       
       
       
       
       
       
       
     Находим примерно такое:

  
  
  
  
  
  
  
В html коде видно, что в блоках с классом «**custom-poster**», лежат нужные нам ссылки на картинки (в теге **img** у аттрибута **src**).

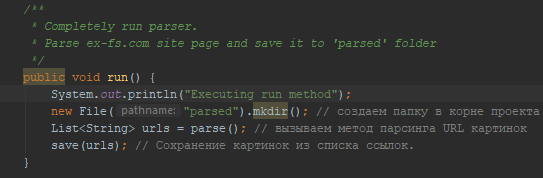
* 1. Класс Elements представляет собой список этих элементов на странице. Поэтому можно проитерироваться по этому списку для получения нужных картинок.  
     Пишем цикл for:  
       
     for (Element link : links) {  
      String imgUrl = url + link.attr("src");  
      result.add(imgUrl); // добавляем URL картинки к результатам  
     }
  2. Для вывода ссылок в консоль добавляем еще один “sout” в конец тела цикла (после result.add(…)):  
     System.out.println(imgUrl);
  3. Наш текущий метод должен возвращать список URL на картинки, для этого корректируем возвращаемое значение: заменяем “return null;”  
     на “return result;”
  4. Запускаем приложения, видим в консоле список ссылок на картинки. Можно открыть парочку.



1. Теперь мы умеем парсить страницу сайта с картинками, теперь попробуем сохранять эти картинки себе на диск.
   1. Метод **save** должен уметь сохранять картинки из полученных URL. Для того чтобы работать с этими ссылками на картинки в этом методе нужно пробросить их в качестве параметра. Добавим параметр в метод:  
      private void save(List<String> urls) { … }
   2. Сохранять мы можем только по одной картинке, поэтому необходимо создать цикл для итерирования по нашему списку. Примечание: еще одно сокращение от IDEA – **fori** .   
      Эта команда создает макет цикла for, необходимо его только немного модифицировать:  
      for (int i = 0; i < urls.size(); i++) {  
       // todo  
      }
   3. Начинаем заполнять тело цикла. Для работы с URL в Java есть одноименный класс:  
       String imgUrl = urls.get(i);  
       URL url = new URL(imgUrl);  
        
      Где urls.get(i) – получение i-го элемента списка, т е наша строка с URL картинки.
   4. Создание объекта URL требует обработки исключения. Оборачиваем в try…catch, аналогично п.16.d.
   5. После создания URL, добавляем строку для чтения картинки, по URL:  
      BufferedImage image = ImageIO.read(url);  
        
      Примечание продолжаем писать в текущем блоке try…catch.
   6. Метод read(url) также требует обработки ошибки, но так как вызов этого метода уже находится в блоке try…catch IDE позволяет расширить текущий блок обработки:  
        
      После этого блок catch немного изменится.
   7. Следующим шагом будет подготовка названия типа будущей картинки:  
        
      String imgType = imgUrl.substring(imgUrl.lastIndexOf('.') + 1);  
        
      Где метод substring возвращает часть строки от начала указанного номера символа и до конца, а imgUrl.lastIndexOf('.') возвращает номер последнего символа ‘.’ в строке.  
      Таким образом мы получаем тип картинки, в нашем случае это jpg.
   8. Следующим шагом будет сохранение картинки с новым именем:  
        
      ImageIO.write(image, imgType, new File("parsed/img\_" +i+ '.' + imgType));  
      Где:  
      image – «байты» нашей картинки  
      imgType – полученный ранее тип картинки  
      new File("parsed/img\_" + i + '.' + imgType) – создание нового файла в папке parsed с новым именем.

Метод save();  


1. Теперь поправим метод run().
   1. Ранее у нас не было параметра в этом методе, после его добавления Java показывает ошибку на вызове этого метода.  
      Поэтому в него нужно передать нужный список, а именно такой список возвращает предыдущий метод parse(). То есть можно передать в метод save() результат метода parse():  
        
      List<String> urls = parse();  
      save(urls);  
      или вообще:  
      save(parse());
   2. Мы сохраняем картинки в папку parsed, но такой папки у нас нет, и если попробуем запустить приложение - получим ошибку, поэтому необходимо создать эту папку.  
      new File("parsed").mkdir();  
        
      Причем создать папку нужно до вызова метода save.  
        
      Метод run



1. Запускаем приложение и проверяем работу парсера.