Clean Code

Chapter 4: Formatting

Author: Hamed Damirchi
hameddamirchi32@gmail.com
github.com/hamed98
linkedin.com/in/hamed-damirchi-ba4085178/

Author: Hamed Damirchi

مقدمه

ما میخوایم وقتی افراد به کد ما نگاه میکنن، تحت تاثیر تمیز بودن کدمون قرار بگیرن!

ما میخوایم اونا این حس رو داشته باشن که این کد توسط حرفه ای ها نوشته شده.

باید به فرمتینگ کد اهمیت داده بشه

باید یه سری قواعد فرمتینگ رو انتخاب کنی و همیشه به اون استاندارد ها پایبند بمونی

اگر یک تیم هستید، حتما باید تیم یک استاندارد فرمتینگ رو انتخاب کنه و همه بهش پایبند بمونن.

Author: Hamed Damirchi

هدف فرمتینگ کد

قبل از هر چیز باید بدونید که فرمتینگ خیلی مهمه!

چون به ارتباط بین دولوپر ها مرتبط میشه و ارتباط، اولین اصل دولوپر های حرفه ایه.

کدی که شما امروز مینویسید ممکنه فردا لازم باشه که تغییرش بدید

استایل و فرمتینگ کد شما تاثیر خیلی زیادی روی کدی که بعدا میخواید بنویسید داره

Author: Hamed Damirchi

فرمتینگ عمودی

یه فایل چقدر باید بزرگ باشه؟

فایل های سورس کد های پروژه های متن باز جاوا چقدر بزرگن؟

صفحه بعد تو نمودار نشون داده شده

مثلا میانگین تعداد خط ها تو fitness، ۶۵ خطه.

بزرگترین فایل تو fitness ۴۰۰ خطه

کوچکترین فایل ۶ خط

*: نمودار تو مقیاس لگاریتمیه

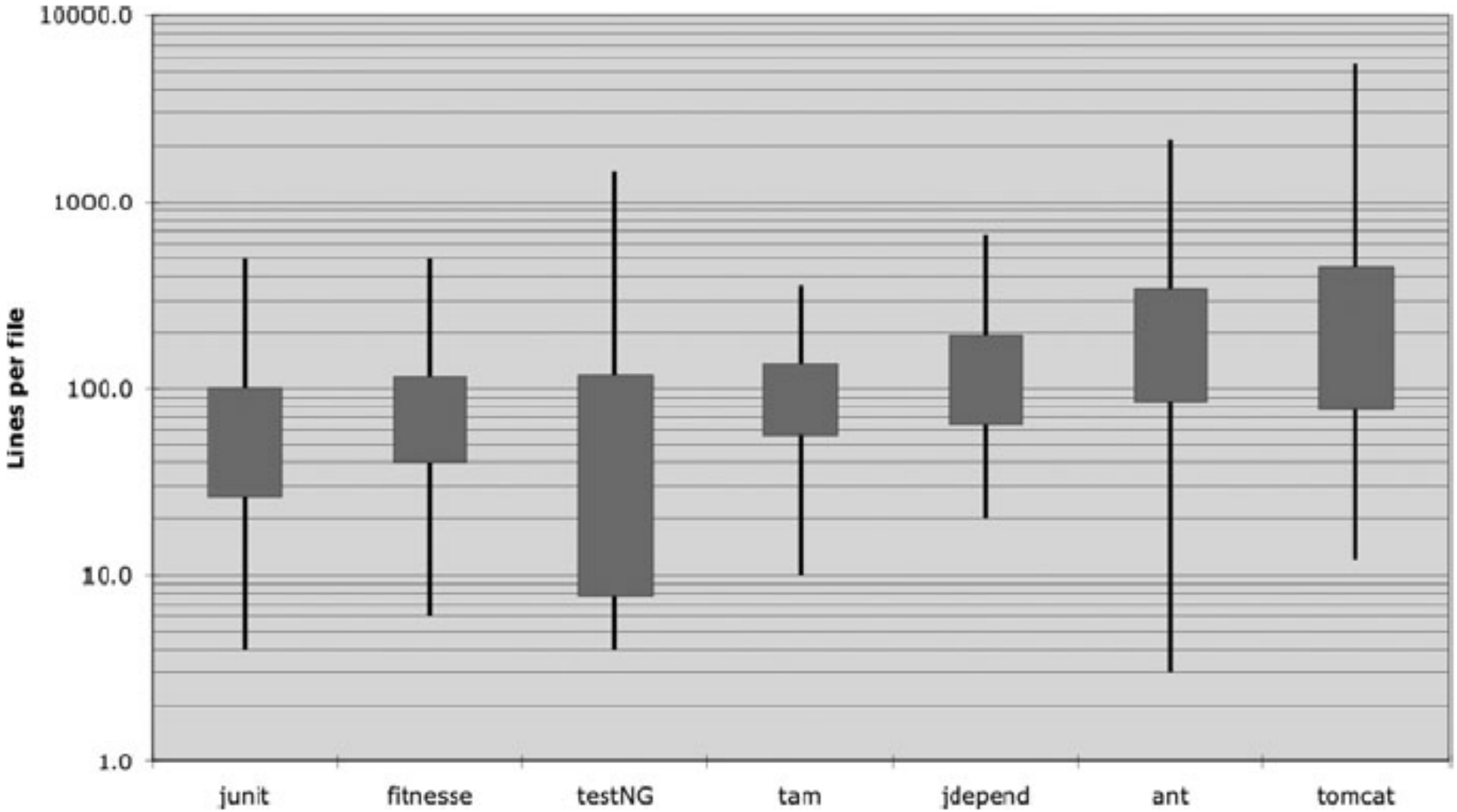

Figure 5-1

File length distributions LOG scale (box height = sigma)

Author: Hamed Damirchi

فرمتینگ عمودی

پروژه های junit,FitNess,Time,Money از فایل های کوچک تری تشکیل شدن. بزرگترین فایل ۵۰۰ خط و به طور متوسط فایل ها از ۲۰۰ خط کمترن

فایل های کوچک تر معمولا خواناتر از فایل های بزرگ هستن.

مطالب:

- **مثل روزنامه**
- **گپ بین گروه کد ها**
- **تراکم عمودی**
- **فاصله عمودی**
- **ترتیب عمودی**

Author: Hamed Damirchi

مثل روزنامه!

به یه روزنامه ای که خوب نوشته شده فکر کنید!

شما اون روزنامه رو عمودی میخونید

انتظار دارید عنوان بالای مقاله، گویای دقیق محتوای نوشته شده باشه و با خوندن عنوان بفهمی که آیا این همون چیزیه که مخیوای بخونی یا نه.

این روزنامه از بالا به پایین که خونده میشه، جزئیات هم بیشتر میشه. یعنی اول عنوان بعد پاراگراف اول که توضیح بیشتر و بعد متن مقاله و بعد اسم نویسنده، تاریخ و ... .

فایل ما هم باید همینطور باشه. اسم فایل در حکم عنوان مقاله اس، باید ساده و در عین حال رسا باشه.

Author: Hamed Damirchi

مثل روزنامه!

ما باید بعد از خوندن اسم یه فایل بفهمیم این فایل همون فایلیه که ما دنبالشیم، یا نه.

بخش های بالایی فایل باید شامل اطلاعات سطح بالا راجع به محتوای فایل(الگوریتمی که نوشته شده) باشه و همینطور که اسکرول میکنیم پایین تر باید سطح انتزاع کمتر شه. یعنی بریم به سطح های پایین تر و جزئیات بیشتر.

یه روزنامه از مقاله های متنوعی تشکیل شده. بعضی هاشون کوچیک، بعضیاشون بزرگن و این باعث میشه روزنامه قابل خواندن بشه اگر روزنامه تنهای از یه داستان طولانی تشکیل شده بود قابل خواندن نبود

Author: Hamed Damirchi

گپ بین گروه کد ها

معمولا هر چند خط کد یه گروهی رو تشکیل میدن که با هم معنی دارن. این گروه باید با گپ(خط خالی) از گروه بعدی متمایز بشه. دو مثال از رعایت این نکته ساده و عدم رعایت اون اورده شده و به نظرم هیچ توضیح اضافه ای لازم نیست و همه چی واضحه که در صورت عدم رعایت این نکته ساده چه قدر کد غیر قابل خواندن میشه.

Author: Hamed Damirchi

مثال رعایت گپ بین گروه کد ها

```
package fitnesse.wikitext.widgets;

import java.util.regex.*;

public class BoldWidget extends ParentWidget {
  public static final String REGEXP = "'''.+?'''";
  private static final Pattern pattern = Pattern.compile("'''(.+?)'''",
    Pattern.MULTILINE + Pattern.DOTALL
  );

  public BoldWidget(ParentWidget parent, String text) throws Exception {
    super(parent);
    Matcher match = pattern.matcher(text);
    match.find();
    addChildWidgets(match.group(1));
  }

  public String render() throws Exception {
    StringBuffer html = new StringBuffer("<b>");
    html.append(childHtml()).append("</b>");
    return html.toString();
  }
}
```

Author: Hamed Damirchi

مثال عدم رعایت گپ بین گروه کد ها

```
package fitnesse.wikitext.widgets;
import java.util.regex.*;
public class BoldWidget extends ParentWidget {
  public static final String REGEXP = "'''.+?'''";
  private static final Pattern pattern = Pattern.compile("'''(.+?)'''",
    Pattern.MULTILINE + Pattern.DOTALL);
  public BoldWidget(ParentWidget parent, String text) throws Exception {
    super(parent);
    Matcher match = pattern.matcher(text);
    match.find();
    addChildWidgets(match.group(1));}
  public String render() throws Exception {
    StringBuffer html = new StringBuffer("<b>");
    html.append(childHtml()).append("</b>");
    return html.toString();
  }
}
```

Author: Hamed Damirchi

تراکم عمودی

این قاعده میگه خط هایی که به هم مرتبطن باید نزدیک هم باشن و متراکم باشن.

مثالی که توش استفاده از کامنت های بی جا باعث شده این قاعده نقض شه و همه چی شلوغ دیده شه:

```
public class ReporterConfig {

  /**
   * The class name of the reporter listener
   */
  private String m_className;

  /**
   * The properties of the reporter listener
   */
  private List<Property> m_properties = new ArrayList<Property>();

  public void addProperty(Property property) {
    m_properties.add(property);
  }
```

تراکم عمودی

در حالی که این قطعه کد خیلی بهتره و انرژی کمتری واسه خوندنش لازمه، سر و چشم برای خوندنش لازم نیست خیلی تکون بخوره

```
public class ReporterConfig {
  private String m_className;
  private List<Property> m_properties = new ArrayList<Property>();

  public void addProperty(Property property) {
    m_properties.add(property);
  }
```

با یه نگاه به کد میشه فهمید که یه کلاسه که دو تا پراپرتی و یه تابع داره. در حالی که تو مثال قبلی به راحتی نمیشه اینو فهمید

Author: Hamed Damirchi

فاصله عمودی

مفاهیمی که به هم دیگه ربط دارن باید نزدیک هم باشن (تو یه فایل)

مفاهیمی که به هم دیگه مرتبط هستن باید تو یه فایل باشن، مگر این که دلیل موجهی واسه جدا کردنشون وجود داشته باشه.

مفاهیمی که تو یه فایل هستن و به هم دیگه ربط دارن، میزان فاصله بینشون یه معیاریه واسه این که بفهمیم چقدر این مفاهیم به هم نزدیکن و چقدر همدیگه رو پوشش میدن (یعنی چقدر به خوانایی همدیگه کمک میکنن)

Author: Hamed Damirchi

فاصله عمودی- تعریف متغیر

متغیرها باید در نزدیک ترین جای ممکن به اون جایی که استفاده شدن تعریف بشن.

در تابع ها، چون تابع های ما خیلی کوچیکن (طبق فصل ۳) متغیر ها بهتره اول تابع تعریف بشن.

```
private static void readPreferences() {
   InputStream is= null;
   try {
      is= new FileInputStream(getPreferencesFile());
      setPreferences(new Properties(getPreferences()));
      getPreferences().load(is);
   } catch (IOException e) {
      try {
         if (is != null)
            is.close();
      } catch (IOException e1) {
      }
   }
}
```

Author: Hamed Damirchi

فاصله عمودی- تعریف متغیر-ادامه

متغیر های حلقه بهتره تو خود حلقه تعریف شن:

```
public int countTestCases() {
   int count= 0;
   for (Test each : tests)
      count += each.countTestCases();
   return count;
}
```

Author: Hamed Damirchi

متغیر های کلاس (instance variables)

این متغیر ها باید اول کلاس تعریف بشن (بالای کلاس)

اما مثلا در سی پلاس پلاس معمولا در انتهای کلاس تعریف میشن

چیزی که مهمه اینه که همشون یه جایی تعریف بشن که هر کسی بتونه راحت محل متغیرو رو پیدا کنه.

صفحه بعد یه مثالیه که این قاعده رعایت نشده و یهویی! دو تا متغیر وسط کلاس تعریف شدن!

Author: Hamed Damirchi

```
public class TestSuite implements Test {
   static public Test createTest(Class<? extends TestCase> theClass,
                                 String name) {
      ...
   }

   public static Constructor<? extends TestCase>
   getTestConstructor(Class<? extends TestCase> theClass)
   throws NoSuchMethodException {
      ...
   }

   public static Test warning(final String message) {
      ...
   }

   private static String exceptionToString(Throwable t) {
      ...
   }

   private String fName;

   private Vector<Test> fTests= new Vector<Test>(10);

   public TestSuite() {
   }

    public TestSuite(final Class<? extends TestCase> theClass) {
      ...
   }

   public TestSuite(Class<? extends TestCase>  theClass, String name) {
      ...
   }
   ... ... ... ... ...
}
```

توابع مرتبط به هم

اگه یه تابع، تابع دیگه ای رو صدا میزنه (هر دو تو یه فایل) این دو تابع باید نزدیک هم باشن و تابع کال کننده باید بالاتر از تابعی که کال شده قرار بگیره. هر چقدر هم این فاصله کمتر باشه بهتره.

این باعث میشه فرایند خوندن کد خیلی راحت تر بشه و تابع کال شونده خیلی راحت تر پیدا میشه. بنابراین خوندن کد سریع تر هم میشه.

یه مثال خوب صفحه ۸۲ کتاب (۱۱۳ پی دی اف) اومده (Listing 5-5)

قرابت مفهومی

هر چقدر کارهایی که دو تابع میکنن، به هم مرتبط تر باشه باید این دو تابع نزدیک تر باشن به هم.

در مثال پایین، این ۴ تابع هم همدیگه رو صدا میزنن و هم کاری که میکنن خیلی شبیه همه، پس حتما باید نزدیک هم باشن

```
public class Assert {
  static public void assertTrue(String message, boolean condition) {
    if (!condition)
      fail(message);
  }

  static public void assertTrue(boolean condition) {
    assertTrue(null, condition);
  }

  static public void assertFalse(String message, boolean condition) {
    assertTrue(message, !condition);
  }

  static public void assertFalse(boolean condition) {
    assertFalse(null, condition);
  }
```

ترتیب عمودی

همونطور که قبلا گفته شد، بهتره تابعی که کال میکنه بالاتر از تابع کال شونده قرار بگیره.

این، شبیه روزنامه ایه که قبلا گفته شد. در روزنامه همینطور که پایین میریم جزئیات بیشتر میشه و وارد سطح های زیرین (سطح انتزاع کمتر) میشیم.

تو کد هم همین شکلیه. وقتی یه تابع، یه تابع دیگه ای رو کال میکنه یعنی تابع اولی یه سطح انتزاع بالاتر از تابع کال شونده است. بنابراین اگر تابع کال شونده پایین تر از کال کننده قرار بگیره، با اسکرول کردن یه فایل به پایین، عملا وارد جزئیات بیشتر و سطوح انتزاع پایین تر میشیم (از مقدار abstract کاسته میشه)

این کار باعث میشه اصطلاحا سرشیر (!) کد رو بگیریم. یعنی با خوندن چند خط اول هر فایل و دیدن تابع هاش، متوجه خلاصه کاری که فایل انجام میده بشیم.

مثال خوب: Listing 15-5 در صفحه ۲۶۳ و Listing 3-7 صفحه ۵۰

فرمتینگ افقی

این نمودار توزیع طول خط ها (بر حسب کاراکتر) در ۷ پروژه مشهور جاوا است:

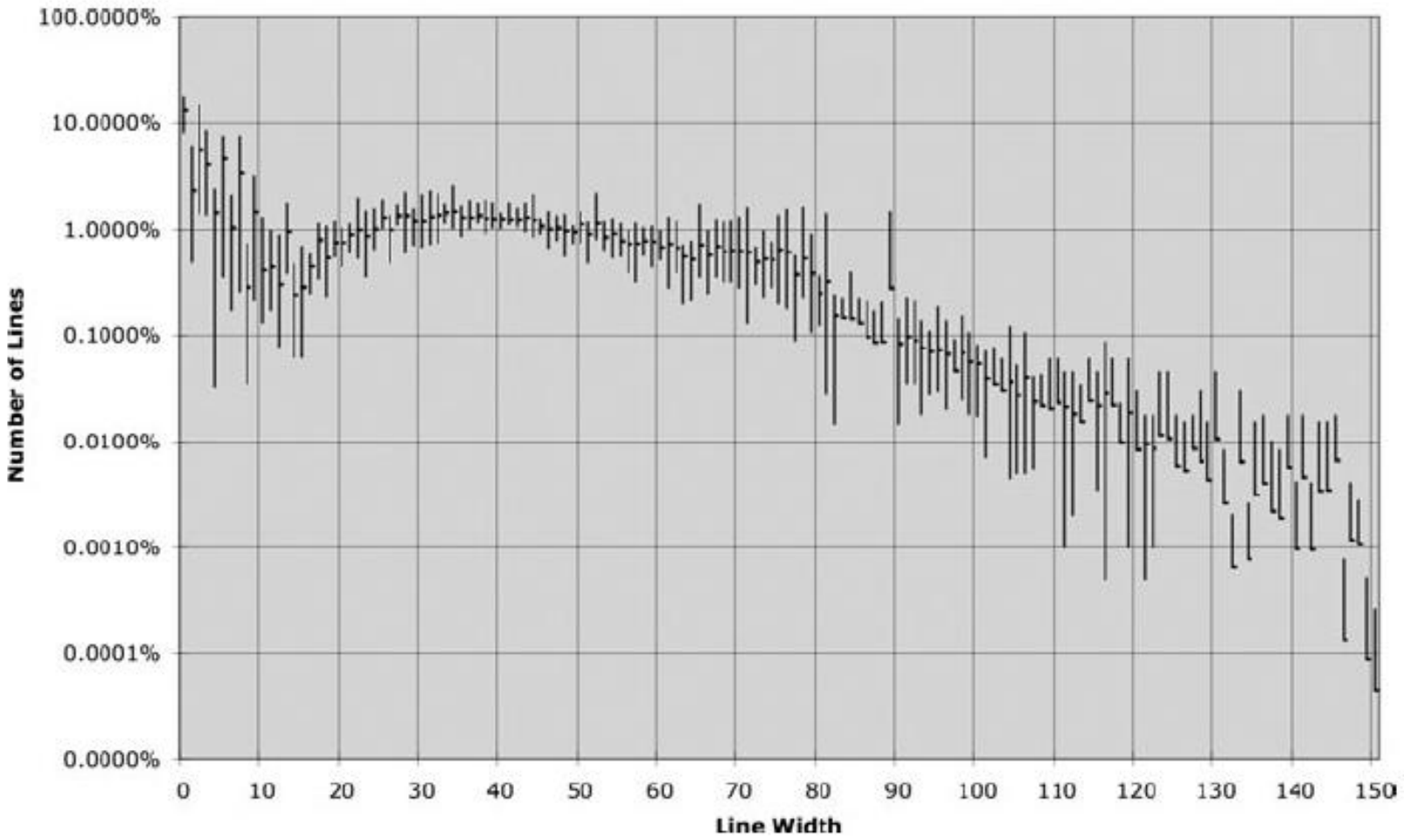

Figure 5-2

Java line width distribution

فرمتینگ افقی-ادامه

طبق نمودار صفحه قبل، حدود ۴۰ درصد از خط های پروژه طول کاراکتری بین ۲۰ تا ۶۰ کاراکتر داشتند.
همچنین ۳۰ درصد از خط ها، کمتر از ۱۰ کاراکتر داشتن.
و خط های بیشتر از ۸۰ کاراکتر خیلی کم هستن.
این یعنی این که طول هر خط باید تا جای ممکن کم باشه
نویسنده میگه من شخصا لیمیت ۱۲۰ کاراکتر رو گذاشتم و به هیچ وجه نباید خطی بیشتر از ۱۲۰ کاراکتر بشه .

مطالب:

- **چگالی افقی**
- **تراز افقی**
- **indention**
- **اسکوپ های ساختگی (dummy scopes)**

Author: Hamed Damirchi

چگالی افقی

مقدار فاصله افقی نشان دهنده میزان ربط داشتن دو بخش مختلفه.

```
private void measureLine(String line) {
  lineCount++;
  int lineSize = line.length();
  totalChars += lineSize;
  lineWidthHistogram.addLine(lineSize, lineCount);
  recordWidestLine(lineSize);
}
```

مثلا تو این کد، اطراف = فاصله گذاشته شده برای این که این اپراتور برجسته بشه و این که سمت چپ و راست این اپراتور دو بخش متفاوت هستن.

از طرفی، بعد از اسم تابع و پرانتز، فاصله گذاشته نشده چون اسم تابع و آرگومان هاش کاملا به هم مرتبط هستن و جداکردنشون اشتباهه چون نشون دهنده تمایزه.

همچنین موقع کال کردن تابع، کنار آرگومان ها هم فصاله گذاشته شده که تمایز بین آرگومان ها رو برسونه و همچنین کاراکتر کاما برجسته تر بشه.

چگالی افقی - ادامه

استفاده دیگه فاصله افقی، برای نشان دادن تقدم عملگر هاست. (اولویت ضرب بیشتر از جمع و ...)

البته خیلی از ابزار های فرمتینگ خودکار از این مورد پشتیبانی نمیکنن و این باعث میشه موقع فرمت کردن این تقدم ها به هم بریزه.

```
public class Quadratic {
  public static double root1(double a, double b, double c) {
    double determinant = determinant(a, b, c);
    return (-b + Math.sqrt(determinant)) / (2*a);
  }

  public static double root2(int a, int b, int c) {
    double determinant = determinant(a, b, c);
    return (-b - Math.sqrt(determinant)) / (2*a);
  }

  private static double determinant(double a, double b, double c) {
    return b*b - 4*a*c;
  }
}
```

Author: Hamed Damirchi

تراز افقی (Horizontal Alignment)

نویسنده میگه من قبلا سعی میکردم این طوری تراز کنم:

```
public class FitNesseExpediter implements ResponseSender
{
  private   Socket          socket;
  private   InputStream     input;
  private   OutputStream    output;
  private   Request         request;
  private   Response        response;
  private   FitNesseContext context;
  protected long            requestParsingTimeLimit;
  private   long            requestProgress;
  private   long            requestParsingDeadline;
  private   boolean         hasError;

  public FitNesseExpediter(Socket          s,
                           FitNesseContext context) throws Exception
  {
    this.context =            context;
    socket =                  s;
    input =                   s.getInputStream();
    output =                  s.getOutputStream();
    requestParsingTimeLimit = 10000;
  }
```

تراز افقی – ادامه

میگه اما بعدا فهمیدم این طور تراز کردن خوب نیست

چون این طور تراز کردن باعث میشه روی چیز های غلطی تاکید بشه

مثلا باعث میشه شما ترغیب بشید اسم متغیر ها رو بخونید، بدون این که به نوع اون متغیر دقت کنید.

یا در بخش بعدی، شما متغیر رو ببینید بدون این که به عملگر = دقت کنید.

همچنین فرمتر های خودکار از این نوع تراز پشتیبانی نمیکنن!

اما نویسنده میگه این روز ها از این نوع تراز استفاده نمیکنم و تمرکزم رو اینه که تعداد کاراکتر های خط کمتر بشه، تراز شدن یا نشدن مهم نیست.

```
public class FitNesseExpediter implements ResponseSender
{
  private Socket socket;
  private InputStream input;
  private OutputStream output;
  private Request request;
  private Response response;
  private FitNesseContext context;
  protected long requestParsingTimeLimit;
  private long requestProgress;
  private long requestParsingDeadline;
  private boolean hasError;

  public FitNesseExpediter(Socket s, FitNesseContext context) throws Exception
  {
    this.context = context;
    socket = s;
    input = s.getInputStream();
    output = s.getOutputStream();
    requestParsingTimeLimit = 10000;
  }
```

Author: Hamed Damirchi

Indention

هر اسکوپی محدوده خودش و متغیر های خودش رو داره. مثلا اسکوپ تابع، کلاس، فایل. این محدوده ها به صورت سلسله مراتبی هستند.

برای این که این سلسله مراتب بهتر دیده بشه، از ایندنت ها استفاده میشه. به این صورت:

- هر چیزی که در خود فایل تعریف میشه، بدون ایندنت (مثلا کلاسی که تو خود فایل تعریف شده)
- تعریف توابع درون کلاس با یک ایندنت نسبت به کلاس
- تعریف بدنه تابع با یک ایندنت نسبت به اسم تابع
- بلوک های داخل تابع هم یک ایندنت میگیرن

این ایندنت گذاری بسیار در خوانایی کد موثره.

Author: Hamed Damirchi

Indention – ادامه

مقایسه کنید:

```
public class FitNesseServer implements SocketServer { private FitNesseContext
context; public FitNesseServer(FitNesseContext context) { this.context =
context; } public void serve(Socket s) { serve(s, 10000); } public void
serve(Socket s, long requestTimeout) { try { FitNesseExpediter sender = new
FitNesseExpediter(s, context);
sender.setRequestParsingTimeLimit(requestTimeout); sender.start(); }
catch(Exception e) { e.printStackTrace(); } } }
```

با کد معادل صفحه بعد:

ادامه – Indention

تو این کد صرفا چند ثانیه وقت میبره تا بفهمید این کد یه فرانته که با سوکت به بک وصل شده

```
public class FitNesseServer implements SocketServer {
  private FitNesseContext context;
  public FitNesseServer(FitNesseContext context) {
    this.context = context;
  }

  public void serve(Socket s) {
    serve(s, 10000);
  }

  public void serve(Socket s, long requestTimeout) {
    try {
      FitNesseExpediter sender = new FitNesseExpediter(s, context);
      sender.setRequestParsingTimeLimit(requestTimeout);
      sender.start();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

Author: Hamed Damirchi

Indention – Breaking Indention

خیلی وقتا وسوسه میشیم که تو قطعه کد های کوچیک (مثلا if ها یا while های یک خطی) ایندنت رو رعایت نکنیم. اما نظر نویسنده اینه که همیشه حتی تو این جور موارد رعایت کنیم.

مثلا به جای این که این شکلی بنویسیم:

```
public class CommentWidget extends TextWidget
{
  public static final String REGEXP = "^#[^\r\n]*(?:(?:\r\n)|\n|\r)?";

  public CommentWidget(ParentWidget parent, String text){super(parent, text);}
  public String render() throws Exception {return ""; }
}
```

این شکلی بنویسیم و ایندنت ها رو همیشه رعایت کنیم:

```
public class CommentWidget extends TextWidget {
  public static final String REGEXP = "^#[^\r\n]*(?:(?:\r\n)|\n|\r)?";

  public CommentWidget(ParentWidget parent, String text) {
    super(parent, text);
  }

  public String render() throws Exception {
    return "";
  }
}
```

اسکوپ های مصنوعی

گاهی اوقات لازم میشه از یه حلقه بدون بدنه استفاده بشه. البته نویسنده ترجیح میده که تا جایی که میشه از این نوع حلقه ها استفاده نکنه، اما اگر لازم شد حتما یه بدنه (اسکوپ) ساختگی واسش بزاره که اشتباه نشه.

نویسنده میگه این موضوع خیلی وقت ها باعث اشتباهش شده چون سمیکالن آخر حلقه for یا while بدون بدنه رو ندیده.

اگر میخواید از بدنه خالی هم استفاده نکنید، لااقل سمیکالن رو بزارید **خط بعدی** که چشم راحت ببینه اونو.

یعنی این طوری:

```
while (dis.read(buf, 0, readBufferSize) != -1)
  ;
```

حرف تیم مهمه!

هر برنامه نویسی قوانین فرمتینگ خودش رو دوست داره. اما مهمه که وقتی تو تیم کار میکنید از قوانین تیم پیروی کنید.

هر تیمی باید قبل شروع به یه قانون فرمتینگ برسه و همه حتما از اون قانون تبعیت کنن.

در آخر هم فرمتینگی که عمو باب ازش استفاده میکنه تو صفحه ۹۱ کتاب (۱۲۲ پی دی اف) اومده.