1.4 Java command-line instructions

We can run, compile, and debug a Java file with command-line instructions. To get the options for any of the commands, one types '-help' soon after the command.

1.4.1 Getting started

Open the shell environment in Mac run Terminal.app in Windows run cmd.exe

To get where is the JDK in your file system from anywhere:

```
> which java
/usr/bin/java
```

To get the version of the JDK from anywhere:

```
>  java -version
java version "1.8.0_25"
Java(TM) SE Runtime Environment (build 1.8.0_25-b17)
Java HotSpot(TM) 64-Bit Server VM (build 25.25-b02, mixed mode)
```

In Mac, Linux, Solaris:
To display the environment variables

```
> env
```

To see the variables in the PATH
In Mac:

```
> echo $PATH
```

In Windows:

```
> PATH
```

To set your PATH to include the JDK sub-folder named java.
In Mac:

```
export PATH=$PATH:/usr/java/jdk1.6.0_10/bin
```

In Windows: For bash, edit the startup file (/.bashrc):

```
PATH=/usr/local/jdk1.7.0/bin:$PATH
```

```
export PATH
```

See also
https://docs.oracle.com/javase/tutorial/essential/environment/paths.html

⊙ **Note:** If you do not set the PATH variable, you need to specify the full path to the executable every time you run it, such as:

```
/usr/bin/javac ~/Unibz/.../NewLectures/LECT2/Dates.java
```

1.4.2 Compilation: javac

The Java Compiler translates programs written in the Java into bytecode.

⊙ **Example:** with the file 'MyClass.java':

```
javac MyClass.java
```

If the system cannot find javac, check the set path command. If javac runs but it returns errors, check the Java text. If the program compiles, but you get an exception, check the spelling and capitalisation in the file name and the class name as Java is case-sensitive.

⊙ **Example:** Where are my .class files? java creates files in the same folder of the .java files. To specify the path javac uses to look up classes needed to run or being referenced by other classes, invoke:

```
> javac -classpath .;C:/users/dac/classes;C:/tools/java/classes ...
```

1.4.3 Execution: java

The java command executes Java class files created by a Java compiler (e.g., java).

⊙ **Example:** Example with the file 'MyClass.class':

```
> java MyClass
```

If the class has main arguments the command is (otherwise

```
> java MyClass 1 2 3 4
```

1.4.4 Disassembler: javap

One can get the bytecode information contained in a .class file with the command line "javap" as in the following foo.java file:

```
[1] class foo{
[2]     void for99(){
[3]         for(int i=0; i<99; i++){}
[4]     }
[5]     void while99(){
[6]         int i =0;
[7]         while(i<99){
[8]             i++;
[9]         }
[10]     }
[11] }
```

```
> javac foo.java
> javap foo.class
```

```
Compiled from "foo.java"
class foo {
  foo();
  void for99();
  void while99();
}
```

or more verbose:

```
> javap -c foo.class
```

```
Compiled from "foo.java"
class foo {
  foo();
    Code:
       0: aload_0
       1: invokespecial #1 // Method java/lang/Object."<init>":()V
       4: return

  void for99();
    Code:
       0: iconst_0
       1: istore_1
       2: iload_1
       3: bipush        99
       5: if_icmpge     14
       8: iinc          1, 1
      11: goto          2
      14: return

  void while99();
    Code:
       0: iconst_0
       1: istore_1
       2: iload_1
       3: bipush        99
       5: if_icmpge     14
       8: iinc          1, 1
      11: goto          2
      14: return
}
```

⊙ **Note:** It is useful to inspect the output of java. For example, we can see that the two loops are the same in the byte code!

```
> javap Analyzer.class
Compiled from "Analyzer.java"
public abstract class org.apache.lucene.analysis.Analyzer {
  protected boolean overridesTokenStreamMethod;
  static java.lang.Class class$java$lang$String;
  static java.lang.Class class$java$io$Reader;
  public org.apache.lucene.analysis.Analyzer();
  public abstract org.apache.lucene.analysis.TokenStream
      tokenStream(java.lang.String, java.io.Reader);
  public org.apache.lucene.analysis.TokenStream
      reusableTokenStream(java.lang.String, java.io.Reader) throws
      java.io.IOException;
  protected java.lang.Object getPreviousTokenStream();
  protected void setPreviousTokenStream(java.lang.Object);
  protected void setOverridesTokenStreamMethod(java.lang.Class);
  public int getPositionIncrementGap(java.lang.String);
  public int getOffsetGap(org.apache.lucene.document.Fieldable);
  public void close();
  static java.lang.Class class$(java.lang.String);
}
```

The javap command disassembles a class file. Its output depends on the options used. If no options are used, javap prints out the public fields and methods of the classes passed to it and prints its output to stdout (standard output).

With the command

```
javap -c Analyzer
```

one gets the full description of the class file by method:

```
public abstract class org.apache.lucene.analysis.Analyzer
    implements java.io.Closeable {
  static final boolean $assertionsDisabled;

  protected org.apache.lucene.analysis.Analyzer();
    Code:
       0: aload_0
       1: invokespecial #1 // Method java/lang/Object."<init>":()V
       4: aload_0
       5: new           #2   // class
           org/apache/lucene/util/CloseableThreadLocal
       8: dup
       9: invokespecial #3  // Method
           org/apache/lucene/util/CloseableThreadLocal."<init>":()V
      12: putfield      #4  // Field
          tokenStreams:Lorg/apache/lucene/util/CloseableThreadLocal;
      15: getstatic     #5  // Field $assertionsDisabled:Z
      18: ifne          36
      21: aload_0
      22: invokespecial #6 // Method assertFinal:()Z
      25: ifne          36
      28: new           #7  // class java/lang/AssertionError
      31: dup
      32: invokespecial #8  // Method
          java/lang/AssertionError."<init>":()V
      35: athrow
      36: return

  public abstract org.apache.lucene.analysis.TokenStream
      tokenStream(java.lang.String, java.io.Reader);

  public org.apache.lucene.analysis.TokenStream
      reusableTokenStream(java.lang.String, java.io.Reader) throws
      java.io.IOException;
    Code:
       0: aload_0
       1: aload_1
       2: aload_2
       3: invokevirtual #24 // Method tokenStream:...;
       6: areturn

  protected java.lang.Object getPreviousTokenStream();
    Code:
       0: aload_0
```

```
    1: getfield     #4    // Field
        tokenStreams:Lorg/apache/lucene/util/CloseableThreadLocal;
    4: invokevirtual #25 // Method
        org/apache/lucene/util/CloseableThreadLocal.get:()Ljava/lang/Object;
    7: areturn
    8: astore_1
    9: aload_0
   10: getfield     #4   // Field
       tokenStreams:Lorg/apache/lucene/util/CloseableThreadLocal;
   13: ifnonnull    26
   16: new          #27  // class
       org/apache/lucene/store/AlreadyClosedException
   19: dup
   20: ldc          #28   // String this Analyzer is closed
   22: invokespecial #29 // Method
       org/apache/lucene/store/AlreadyClosedException."<init>":(Ljava/lang/String;)V
   25: athrow
   26: aload_1
   27: athrow
 Exception table:
    from    to  target type
        0     7     8   Class java/lang/NullPointerException

protected void setPreviousTokenStream(java.lang.Object);
  Code:
    0: aload_0
    1: getfield     #4   // Field
        tokenStreams:Lorg/apache/lucene/util/CloseableThreadLocal;
    4: aload_1
    5: invokevirtual #30 // Method
        org/apache/lucene/util/CloseableThreadLocal.set:(Ljava/lang/Object;)V
    8: goto         31
   11: astore_2
   12: aload_0
   13: getfield     #4   // Field
       tokenStreams:Lorg/apache/lucene/util/CloseableThreadLocal;
   16: ifnonnull    29
   19: new          #27   // class
       org/apache/lucene/store/AlreadyClosedException
   22: dup
   23: ldc          #28  // String this Analyzer is closed
   25: invokespecial #29 // Method
       org/apache/lucene/store/AlreadyClosedException."<init>":(Ljava/lang/String;)V
   28: athrow
   29: aload_2
```

```
    30: athrow
    31: return
  Exception table:
     from    to  target type
         0     8    11   Class java/lang/NullPointerException

public int getPositionIncrementGap(java.lang.String);
  Code:
     0: iconst_0
     1: ireturn

public int getOffsetGap(org.apache.lucene.document.Fieldable);
  Code:
     0: aload_1
     1: invokeinterface #31, 1 // InterfaceMethod
         org/apache/lucene/document/Fieldable.isTokenized:()Z
     6: ifeq        11
     9: iconst_1
    10: ireturn
    11: iconst_0
    12: ireturn

public void close();
  Code:
     0: aload_0
     1: getfield    #4  // Field
         tokenStreams:Lorg/apache/lucene/util/CloseableThreadLocal;
     4: invokevirtual #32 // Method
         org/apache/lucene/util/CloseableThreadLocal.close:()V
     7: aload_0
     8: aconst_null
     9: putfield    #4  // Field
         tokenStreams:Lorg/apache/lucene/util/CloseableThreadLocal;
    12: return

static {};
  Code:
     0: ldc_w       #33  // class
         org/apache/lucene/analysis/Analyzer
     3: invokevirtual #10 // Method
         java/lang/Class.desiredAssertionStatus:()Z
     6: ifne        13
     9: iconst_1
    10: goto        14
    13: iconst_0
```

```
      14: putstatic   #5   // Field $assertionsDisabled:Z
      17: return
}
```

When do I use javap?

Javap is useful when you want to see what your compiler is doing and the effects a code change has on a compiled class file.

⊙ **Example:** StringBuffer vs. String

Below is a contrived class that has two methods that return a String consisting of the numbers 0 to n, where n is supplied by the caller. The only difference between the two methods is that one uses a String to build the result, and the other uses a StringBuffer.

```
public class JavapTip {
    public static void main(String[] args) {
    }
    public static String withStrings(int count) {
        String s = "";
        for (int i = 0; i < count; i++) {
            s += i;
        }
        return s;
    }
    public static String withStringBuffer(int count) {
        StringBuffer sb = new StringBuffer();
        for (int i = 0; i < count; i++) {
            sb.append(i);
        }
        return sb.toString();
    }
}
```

⊙ **Note:** String is immutable and StringBuffer not

Running this

```
>javap -c JavapTip
```

has the output:

```
Compiled from "JavaTip.java"
```

```
public class JavaTip {
  public JavaTip();
    Code:
       0: aload_0
       1: invokespecial #1                  // Method
           java/lang/Object."<init>":()V
       4: return

  public static void main(java.lang.String[]);
    Code:
       0: return

  public static java.lang.String withStrings(int);
    Code:
       0: ldc           #2                  // String
       2: astore_1
       3: iconst_0
       4: istore_2
       5: iload_2
       6: iload_0
       7: if_icmpge     35
      10: new           #3                  // class
          java/lang/StringBuilder
      13: dup
      14: invokespecial #4                  // Method
          java/lang/StringBuilder."<init>":()V
      17: aload_1
      18: invokevirtual #5                  // Method
          java/lang/StringBuilder.append:(Ljava/lang/String;)Ljava/lang/StringBuilder;
      21: iload_2
      22: invokevirtual #6                  // Method
          java/lang/StringBuilder.append:(I)Ljava/lang/StringBuilder;
      25: invokevirtual #7                  // Method
          java/lang/StringBuilder.toString:()Ljava/lang/String;
      28: astore_1
      29: iinc          2, 1
      \textbf{32: goto        5}
      35: aload_1
      36: areturn

  public static java.lang.String withStringBuffer(int);
    Code:
       0: new           #8                  // class
           java/lang/StringBuffer
       3: dup
```

```
       4: invokespecial #9                  // Method
           java/lang/StringBuffer."<init>":()V
       7: astore_1
       8: iconst_0
       9: istore_2
      10: iload_2
      11: iload_0
      12: if_icmpge   27
      15: aload_1
      16: iload_2
      17: invokevirtual #10                 // Method
          java/lang/StringBuffer.append:(I)Ljava/lang/StringBuffer;
      20: pop
      21: iinc        2, 1
       \textbf{24: goto        10}
      27: aload_1
      28: invokevirtual #11                 // Method
          java/lang/StringBuffer.toString:()Ljava/lang/String;
      31: areturn
}
```

You can see that the withStrings() method creates a new StringBilder instance each time through the loop. Then, it appends the current value of the existing String to the StringBilder and appends the current value of the loop. Finally, it calls toString on the buffer and assigns the results to the existing String reference. With the goto the loop starts again at line 5 reinstating a StringBuilder.

This is in contrast to the withStringBuffer method, which only calls the existing StringBuffer's append method each time through the loop. There's no object creation and no new String references. With the goto the loop starts again at line 10 not instating a StringBuffer.

We know why this happen: String is immutable and everytime a new object must be instantiated!

1.4.5 Debugger: jdb

The Java Debugger helps find and fix bugs in Java programs. There are two major categories of errors: compiler errors and execution (or run-time) errors and then there are failures. In the following we will see compiler errors and failures. In the following, we learn how to discover them.

⊙ **Example:** The following file DatesBuggy.java contains some syntax errors[1]:

[1]Thank to prof. Mary K. Vernon http://pages.cs.wisc.edu/ vernon/cs367/tutorials/jdb.tutorial.html

```
1  import java.io.*;
2  class DatesBuggy {
3  public static int daysInMonth (int month) {
4      if (month == 9) || (month == 4) || (month == 6) || (month ==
    11)) {
5           return 30;
6      }
7      else if (month == 2)
8            return 28;
9      else return 31;
10 }
11 public static void main (String[] args) {
12     int someMonth, someDay;
13     int laterMonth, laterDay;
14     int aMonth;
15     someMonth = Integer.parseInt(args[0]);
16     someDay = Integer.parseInt(args[1]);
17     laterMonth = Integer.parseInt(args[2]);
18     laterDay = Integer.parseInt(args[3]);
19     /* Used to record what day in the year the first day */
20     /* of someMonth and laterMonth are. */
21     int someDayInYear = 0;
22     int laterDayInYear = 0;
23     for (aMonth = 0, aMonth < someMonth; aMonth = aMonth + 1) {
24         someDayInYear = someDayInYear + daysInMonth(aMonth);
25     }
26     for (aMonth = 1; aMonth < laterMonth; aMonth = aMonth + 1) {
27         laterDayInYear = laterDayInYear + daysInMonth(aMonth);
28     }
29     /* The answer */
30     int daysBetween = 0;
31     System.out.println("The difference in days between " +
32                        someMonth + "/" + someDay + " and " +
33                        laterMonth + "/" + laterDay + " is: ");
34     daysBetween = laterDayInYear - someDayInYear;
35     daysBetween = daysBetween + laterDay - someDay;
36     System.out.println(daysBetween);
37 }
38 }
```

Compile the file with javac:

```
> javac DatesBuggy.java
DatesBuggy.java:6: error: illegal start of expression
    if (month == 9) || (month == 4) || (month == 6) || (month ==
        11)) {
                   ^
DatesBuggy.java:6: error: not a statement
    if (month == 9) || (month == 4) || (month == 6) || (month ==
        11)) {
                                                  ^
DatesBuggy.java:6: error: ';' expected
    if (month == 9) || (month == 4) || (month == 6) || (month ==
        11)) {
                                                                 ^
DatesBuggy.java:9: error: 'else' without 'if'
    else if (month == 2)
    ^
DatesBuggy.java:29: error: not a statement
    for (aMonth = 0, aMonth < someMonth; aMonth = aMonth + 1) {
                         ^
DatesBuggy.java:29: error: ';' expected
    for (aMonth = 0, aMonth < someMonth; aMonth = aMonth + 1) {
                                                          ^
6 errors
```

Look at the code and notice that:
Compiler Error1: The code lacks a parenthesis in the first if condition.
Compiler Error2: The code lacks a semicolon in the first for condition.

Fix the compiler errors, compile and run it again:

```
>  javac DatesBuggy.java
>  java DatesBuggy 1 12 3 4
The difference in days between 1/12 and 3/4 is:
20
```

Now you do not get and error but wrong answer!
Failure1: number of days is wrong. (Correct output is 51. Where is the error in the code?)
Recompile the program with the '-g' option to tell the compiler to provide information that jdb can use to display local (stack) variables:

```
>javac -g DatesBuggy.java
```

Run your program by having jdb start the Java interpreter:

```
>jdb DatesBuggy 1 12 3 4
```

At this point, jdb has invoked the Java interpreter, the DatesBuggy class is loaded, and the interpreter stops before entering main().
Give the command 'stop in DatesBuggy.main' and then 'run'. The interpreter will continue executing for a very short time until just after it enters main().

```
> jdb DatesBuggy 1 12 3 4
Initializing jdb ...
> stop in DatesBuggy.main
Deferring breakpoint DatesBuggy.main.
It will be set after the class is loaded.
> run
run DatesBuggy 1 12 3 4
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
>
VM Started: Set deferred breakpoint DatesBuggy.main

Breakpoint hit: "thread=main", DatesBuggy.main(), line=18 bci=0
18       someMonth = Integer.parseInt(args[0]);
main[1]
```

Type list to see the source code for the instructions that are about to execute, or you can type print args to see the value of the variable called "args"

```
main[1] list
14       int someMonth, someDay;
15       int laterMonth, laterDay;
16
17       int aMonth;
18 =>    someMonth = Integer.parseInt(args[0]);
19       someDay = Integer.parseInt(args[1]);
20
21       laterMonth = Integer.parseInt(args[2]);
22       laterDay = Integer.parseInt(args[3]);
23
```

One tricky point about jdb is that if you use the step command to execute one instruction at a time, you will step into the instructions for the method

called Integer.parseInt. The source code for predefined Java classes is not available to jdb, so jdb cannot list the lines of code or print the values of any variables in that method. Thus, you should set a breakpoint after the arguments are parsed, using the command stop at DatesBuggy:30 and then type 'cont' to let the interpreter continue executing until it again reaches a breakpoint (i.e., until it is about to execute line 24). You will get null after the command locals or print below if you did not run the compiler java with "-g".

```
> jdb DatesBuggy 1 12 3 4
Initializing jdb ...
> stop at DatesBuggy:24
Deferring breakpoint DatesBuggy:241.
It will be set after the class is loaded.
> run
run DatesBuggy 1 12 3 4
Set uncaught java.lang.Throwable
Set deferred uncaught java.lang.Throwable
>
VM Started: Set deferred breakpoint DatesBuggy:24

Breakpoint hit: "thread=main", DatesBuggy.main(), line=24 bci=44
24          someDayInYear = someDayInYear + daysInMonth(aMonth);

main[1] print laterMonth
laterMonth = 3
main[1] locals
Method arguments:
args = instance of java.lang.String[4] (id=439)
Local variables:
someMonth = 1
someDay = 12
laterMonth = 3
laterDay = 4
someDayInYear = 0
laterDayInYear = 0
aMonth = 0
main[1]
```

One should continue to examine the program's behaviour as it executes by setting further breakpoints, or using **step** to execute one instruction at a time. At each breakpoint, use the print or locals command to examine the values of program variables, until you isolate the error. Note that when the

method called daysInMonth is called, jdb can stop at a breakpoint in that method (e.g., if you say stop in DatesBuggy.daysInMonth) and it can list the code or print variable values in that method. The error in DatesBuggy can be corrected by changing only ONE line.

When you think you have found the error: copy the file to another file in case you need to use it later, correct the error, and recompile and execute it to see if the problem is solved. Type exit to exit the debugging.

1.4.6 javadoc - the Java documentation

```
>javadoc DatesBuggy.class
```

To tell the environment where to put the javadoc files

⊙ **Example:** My project contains the file
ownCloud/workspace/Dates/src/DatesBuggy.java
and I want to put the generated documentation in files located in a specific folder
ownCloud/workspace/Dates/doc

```
>javadoc -d ownCloud/workspace/Dates/doc
    ownCloud/workspace/Dates/src/DatesBuggy.java
```

or

```
> javadoc -d doc src/DatesBuggy.java
```

if you are already in the local folder Dates

In general:

```
> javadoc -d [path to javadoc destination directory] [package name]
```

1.4.7 Packaging Programs in jar files

The jar command compresses your project into a portable file. In the example, you compress into a file called Dates.jar, both a .class and .java file.

```
> jar cf Dates.jar Dates.java Dates.class
```

You can easily send your jar file to anyone. You can put .java and/or .class file

⊙ **Note:** When do you want to include the .java or .class files? Typically you circulate the .class file to protect you source code. The byte code can be executed anywhere

You need to include a manifesto file. What is it?

Including and modifying a MANIFEST.MF file.

The Jar tool automatically puts a default manifest with the pathname META-INF/MANIFEST.MF into any JAR file one creates. The default

MANIFEST.MF file includes the information about where the main class is.

For the class called Dates the default MANIFEST looks like the following

```
Manifest-Version: 1.0
Created-By: 1.8.0_25 (Oracle Corporation)
Main-Class: Dates
```

You can enable special JAR file functionality, such as package sealing, by modifying the default manifest. Typically, modifying the default manifest involves adding special purpose headers to the manifest that allow the JAR file to perform a particular desired function.

⊙ **Example:** MANIFEST.MF file

```
Manifest-Version: 1.0
Ant-Version: Apache Ant 1.8.2
Created-By: 1.6.0-b105 (Sun Microsystems Inc.)

Name: epl-v10.html
SHA1-Digest: hNPRR3joO5UV8+u/L0qjKV/Y2EE=

Name: META-INF/eclipse.inf
SHA1-Digest: KyT9FF7C7t86NoBoa2kZT3ZJBfw=

Name: eclipse_update_120.jpg
SHA1-Digest: xstAqMgs/a5AsQXQZSdDQ79veOA=

Name: license.html
SHA1-Digest: /vLZjlHkZSXMSfPrWwNqOUDqqbM=

Name: feature.properties
SHA1-Digest: hpJCuONFEtZ9bJJKB2dS51Ek1QM=

Name: feature.xml
SHA1-Digest: rXv5ZcgfdBOUa30DcKm2s4D+/o8=
```

To modify the manifest, you must first prepare a text file containing the information you wish to add to the manifest. You then use the jar "m" option to add the information in your file to the manifest. The text file from which you are creating the manifest must end with a new line or carriage return. The last line will not be parsed properly if it does not end with a new line or carriage return. The contents of the manifest must be encoded

in UTF8.

The command to add it is the following:

```
>jar cfm jar-file manifest-addition input-file(s)
```

The c option indicates that you want to create a JAR file.

The m option indicates that you want to merge information from an existing file into the manifest file of the JAR file you're creating.

The f option indicates that you want the output to go to a file (the JAR file you're creating) rather than to standard output.

manifest-addition is the name (or path and name) of the existing text file whose contents you want to add to the contents of JAR file's manifest.

jar-file is the name that you want the resulting JAR file to have.

The input-file(s) argument is a space separated list of one or more files that you want to be placed in your JAR file.

The m and f options must be in the same order as the corresponding arguments.

```
>jar cfm Dates.jar Manifest.txt Dates.class
```

You can also add the source file

```
>jar cfm Dates.jar Manifest.txt Dates.class Dates.java
```

The execute the jar file

```
>java -jar Dates.jar
```

If the .jar file does not contain the .class file you get the error:

```
>java -jar Dates.jar
Error: Could not find or load main class Dates
```

If the jar file does not contain the MANIFEST file you get the error

```
>java -jar Dates.jar
no main manifest attribute, in Dates.jar
```

If you do not pass the arguments of the main when it is required you get the error:

```
>java -jar Dates.jar
Exception in thread "main"
    java.lang.ArrayIndexOutOfBoundsException: 0
```

```
    at Dates.main(Dates.java:15)
```