Recursion and External Packages

Lecture 2 in 1DV507 - Programming and Data Structures

Dr Jonas Lundberg, office B3024

Jonas.Lundberg@lnu.se

January 21, 2019

Agenda

- Recursive Methods
  - Definition and example
  - Solving problems using recursion

- Using External Packages
  - A concrete example: MP3 Player
  - A quick look at Exercise 6, Assignment 1

- Reading Instructions
  Horstmann: Sections 13.1-13.5
  Liang: Chapter 18

Recursion

- What is recursion?
  - A way to think when solving problems.
  - An implementation where a method calls itself.
- Why is it important?
  - The recursive solution is sometimes much easier than the iterative.
  - Examples: Binary search trees, Hanoi Tower, Fibonacci numbers
- Is it always good?
  - No, in many cases an iterative solution is just as good.
  - In certain cases (e.g. Fibonacci) recursion is simple but very costly.
  - We will today see many examples that can just as easily be handled using iteration.
- Is it possible to solve more problems using recursion than without?
  - Speaking purely computationally, no. On the other hand, many problems are much simpler using recursion.

Arithmetic Sum (Recursion intro.)

$$S(n) = \sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + (n-2) + (n-1) + n$$

- Can be computed iteratively:

  ArithmSumIt(n)

  1. $sum = 0$
  2. $count = 0$
  3. Repeat n times:
     3.1 $count = count + 1$
     3.2 $sum = sum + count$
  4. The answer is now in sum

Computing sum using smaller sums

$$S(n) = \sum_{i=1}^{n} i = \underbrace{1 + 2 + 3 + \cdots + (n-2) + (n-1)}_{S(n-1)} + n$$

- The problem can be expressed using a smaller problem: $S(n) = S(n-1) + n$
- **Ex**: S(5) = S(4) + 5
- And moving on ...
  - S(4) = S(3) + 4
  - S(3) = S(2) + 3
  - S(2) = S(1) + 2
  - S(1) = S(0) + 1
  - S(0) = S(-1) + 0 ???

We must find a base case ⇒ a case where it all stops!

Arithmetic Sum: Introducing a Base Case

- We need a *base case* to terminate the computation.
- We choose to set the base case to $S(1) = 1$ ($S(0) = 0$ would also work).
- The base case is expressed as a fact, not referring to any smaller problems.
- We now have a *recursive definition*:

$$S(n) = \begin{cases} 1 & n = 1 \qquad (\textit{base case}) \\ S(n-1) + n & n \geq 2 \qquad (\textit{recursive step}) \end{cases}$$

- $S(4) = S(3) + 4 = S(2) + 3 + 4 = S(1) + 2 + 3 + 4 = 1 + 2 + 3 + 4$

Arithmetic Sum: A Recursive Solution

- From the recursive definition a recursive algorithm to compute $S(n)$ can be constructed:

  ArithmSumRec(n)

  1. If $n = 1$
     1.1 return 1
  2. Else
     2.1 return ArithmSumRec(n-1) + n

Example: Executing `ArithmSumRec(5)`

ArithmSumRec(n)

1. If $n = 1$
   1.1 return 1
2. Else
   2.1 return ArithmSumRec(n-1) + n

Executing `ArithmSumRec(5)` $\Rightarrow$ 5 calls to `ArithmSumRec(...)`

```
ArithmSumRec(5)
   ArithmSumRec(4)
      ArithmSumRec(3)
         ArithmSumRec(2)
            ArithmSumRec(1)
            return 1                                // base case
         return 1 + 2                             (= 3)
      return 3 + 3                              (= 6)
   return 6 + 4                              (= 10)
return 10 + 5                              (= 15)
```

Recursion

- Compute a solution to a problem using a smaller (but similar) problem is called *recursion*.
- In general, recursion $\Rightarrow$ a method calls itself.
- In order not to be trapped in an inifinite loop, a base case (at least one) must be part of the definition.
- Everything that can be done recursively, can also be done iteratively but not always as easy.
- Recursive definitions and algorithms are common in mathematics and computer science.

Factorial: A Recursive Definition

- The factorial, for example 5!, is computed as:

$$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120$$

- The factorial (N!) for a positive integer N can be defined as:

$$N! = \begin{cases} 1 & \text{when } N = 1 \\ N \cdot (N-1)! & \text{when } N > 1 \end{cases}$$

Note

- We use (N-1) factorial to define N-factorial.
  $\Rightarrow$ *recursion*.
- Repeated usage of the definition for N! gives the answer

$$5! = 5 \cdot 4! = 5 \cdot 4 \cdot 3! = 5 \cdot 4 \cdot 3 \cdot 2! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$$

Palindrome: A Recursive Definition

- A string is a *simple palindrome* if it has the same text in reverse.
- Examples: x, anna, madam, abcdefedcba, yyyyyyyy
- A palindrome can be defined as:
  1. An empty string is a palindrome
  2. A string with the length 1 is a palindrome.
  3. A string is a palindrome if the first and last characters are equal, and all characters in between is a palindrome.
- 1 and 2 are our base cases
- 3 is our recursive step

Java: Recursive Method (1)

- A recursive method to compute n-factorial.

```
public int factorial (int n) {
    if (n == 1)
        return 1;
    else
        return n* factorial (n-1);
}
```

Note

- Recursive methods calls themselves in one or several steps.
- Indirect recursion ⇒ call oneself after several steps.

```
public void a(int n) {          public void b(int n) {
    ...                                 ...
    ...                                 ...
    b(7);                               a(23);   // recursion!
    ...                                 ...
}                               }
```

Java: Recursive Method (2)

- A recursive method for checking a palindrome.

```
public boolean isPalindrome(char[] str, int p, int q) {
    if (q <= p)                    // Base case
        return true;
    else if (str[p] != str[q])
        return false;
    else
        return isPalindrome(str,p+1,q-1);
}
```

- Usage:

```
char[] word = "madam".toCharArray();         // Example
boolean b = isPalindrome(word,0,4);           // true

char[] word = ...                             // In general
b = isPalindrome(word,0,word.length-1);       // true or false
```

Recursive Helper Methods

```
public static void main(String[] args) {
    String word = "madam";
    char[] chars = word.toCharArray();
    boolean b = isPalindrome(chars, 0, chars.length-1);

    // Using a helper method
    b = isPalindrome(word);
}

public static boolean isPalindrome(String word) { // Helps to get started
        return isPalindrome( word.toCharArray() , 0, chars.length-1);
}

public static boolean isPalindrome(char[] str, int p, int q) {
    if (q <= p)                 // Base case
        return true;
    else if (str[p] != str[q])
        return false;
    else
        return isPalindrome(str,p+1,q-1);
}
```

Recursive Methods (In General)

- A recursive method consists of:
  - One or more *base cases* where "simple" results are given explicitly.
  - One or more recursive rules (or steps) where "larger" results are expressed using "smaller" results.
- **Note**
  - We use *recursive rules* until a problem has been reduced to size where a base case can be used.
  - No base case $\Rightarrow$ infinite recursion $\Rightarrow$ program will crash.
- **Crash in practice**

```
public static void main(String[] args) {
    m(0);
}

public static int m(int n) {
    return m(n+1);      // no base case ==> infinite recursion
}
```

The program runs for a second and raises a StackOverflowError
Stack: Part of JVM memory keeping track of ongoing calls.

Examples and Exercises

- In the *Fibonacci* sequence the first two numbers are 0 and 1 and the others are the sum of the two previous numbers.

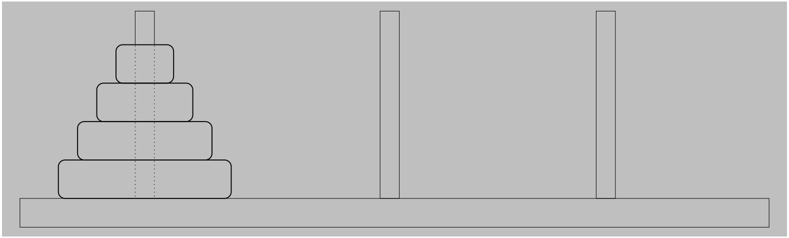
  `0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...`

- **Exercise**: Write a recursive method `int fib(int n)` that computes the n:th number in the Fibonacci sequence. For example `fib(0) = 0`, `fib(1) = 1`, and `fib(6) = 8`.

- **Exercise**: Write a recursive method `int mult(int a, int b)` that computes the multiplication $a \cdot b$ with the use of addition. Assume that both a and b are positive.

- **Exercise**: Write a recursive method `print(int[] a, int n)` printing the content of a:
  - **a)** starting from position 0 and onwards
  - **b)** in reverse order

Towers of Hanoi (Problem)



- **Problem**: Move the discs from one pole to the other.
- Rules:
  - Only one disc can be moved at a time.
  - A larger disc can not be placed on a smaller disc.
  - All discs must be on a pole, except for the one that is moved.
- See *Worked Example 13.2* in Horstmann or Section 18.7 in Liang for more details.

Tower of Hanoi

- Algorithm:
  1. Move N-1 discs from start to help pole.
  2. Move the lowest disc from start to end pole.
  3. Move N-1 discs from help to end pole.
- Usage: `moveTower(6,1,2,3)`
  6 is the number of discs, 1 is start pole, 2 is end pole, and 3 is temp pole
- Solution:

```
void moveTower(int NumDisks, int start, int end, int temp)  {
   if (numDisks == 1)
      System.out.println("Move disc from "+start+" to "+end);
   else {
      moveTower(numDisks-1,start,temp,end);
      System.out.println("Move disc from  "+start+" to "+end);
      moveTower(numDisks-1,temp,end,start);
   }
}
```

- See *Worked Example 13.2* in the book for more details.

DirectoryMain.java

- Problem: List all subdirectories.

```
public static void main(String[] args) {
    File startDir = new File("C:\\undervisning\\DA1021");
    visitSub(startDir);        // Start recursive visit of subdirectories
}
```

```
Printing:
 1  DA1021
 2    kursutv
 3    labbar
 4    lectures
 5      array_list
 6        figures
 7      graphics1
 8        figures
 9      graphics2
10        figures
11      graphics3
12        figures
13      intro
14        figures
```

Visit Subdirectories – Continued

- Visit and print each subdirectory recursively.

```
private static int depth = 1, count = 0;             // Indentation

private static void visitSub(File file) {
    if (file.isDirectory()) {
        printDir(file);
        depth++;                        // Increase before visiting subdirectories
        File[] subs = file.listFiles();
        for (File f : subs)
            visitSub(f);
        depth--;                        // Decrease after visiting subdirectories
    }
}

private static void printDir(File file) {  // Indented printing
    StringBuffer buf = new StringBuffer();
        for (int i=0; i<depth; i++)          // Add indentation
            buf.append("  ");
    System.out.println((++count) + buf.toString() + file.getName());
}
```

The First 100 Fibonacci Numbers

- A recursive method for computing the N:th number in the Fibonacci sequence.

```
public int fib(int N) {
    if (N==0)
        return 0;
    else if (N==1)
        return 1;
    else
        return fib(N-1) + fib(N-2);
}
```

Problem: Print the first 100 numbers in the Fibonacci sequence.

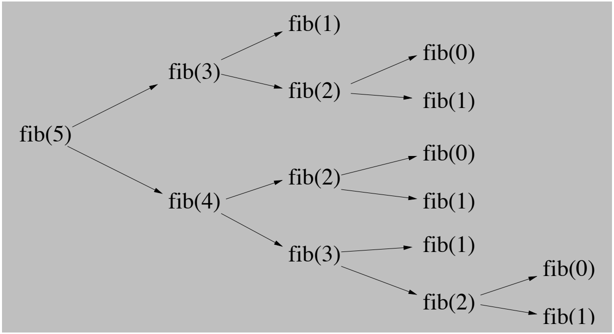
Simple!

```
for (int i=0;i<100;i++)
    System.out.println( fib(i) );
```

Result: The program races and then dies.

Exponential Number of Calls

- Computing `fib(5)`



- fib(5) takes 15 calls to fib(N).
- fib(6) takes 25 calls to fib(N).
- fib(100) takes an enormous amount of calls to fib(N).
- All values between 1 and N $\Rightarrow$ the number is proportional to 2^N $\Rightarrow$ the computer crashes for $N = 100$.

A better solution

```
public static void main(String[] args) {
    int N = 90;                    // N=100 does not work
    long fm2 = 0, fm1 = 1, f;
    for (int i = 3; i<N; i++) {
        f = fm1 + fm2;
        System.out.println(i + "\t" + f);
        fm2 = fm1;
        fm1 = f;
    }
}
```

```
The last five parts
  85   259695496911122585
  86   420196140727489673
  87   679891637638612258
  88   1100087778366101931
  89   1779979416004714189
```

- $N = 100$ does not work, the number is too large for a `long`.

A working solution for N = 100

- With the class `BigInteger` it is possible to use N = 100.

```
public static void fibIterate (int N) {
    BigInteger fm2 = new BigInteger("0");
    BigInteger fm1 = new BigInteger("1");
    BigInteger f = new BigInteger("0");
    for (int i = 2;i<N;i++) {
        f = fm1.add(fm2);
        System.out. println (i+"\t"+f);
        fm2 = fm1;
        fm1 = f;
    }
}
```

- The library class `BigInteger` is designed to handle very large integer numbers

Assignment 1: Three Recursive Exercises

- Exercise 4: Print the N:th line in Pascal's triangle.

linje 0 →							1							
linje 1 →						1		1						
linje 2 →					1		2		1					
linje 3 →				1		3		3		1				
linje 4 →			1		4		6		4		1			
linje 5 →		1		5		10		10		5		1		
linje 6 →	1		6		15		20		15		6		1	

- The program must have a recursive method

```
int[] pascalLine(int order)
```

- that calculates the n:th line in the triangle.

Exercises 2-4 are all about recursion. Not much coding. Tricky if you are not used to recursion.

Closing remarks on recursion

- At first glance recursive solutions to problems can seem difficult.
  - Mainly because they are different than an iterative solution.
- See it as yet another tool that is possible to use for some problems.
  - And as almost the only solution to some problems as we will see when discussing binary search trees.
- It is also important to remember that a recursive solution very seldom runs faster than an iterative – so make careful analysis before using it.

A 10 Minute Break .....

ZZZZZZZZZZZZZZZZZZZZZZZZZZ

Program parameters using `String[] args`

```
package lecture2;

public class ArgsMain {
    public static void main(String[] args) {
        System.out.println("Arguments: "+args.length);
        for (String s : args)
            System.out.println("\t"+s);
    }
}
```

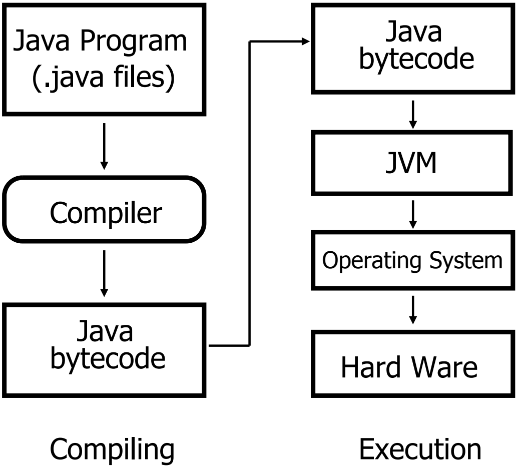
Eclipse

- Open `Run` ⇒ `Open Run Dialog` ⇒ `Arguments`
- Write the program parameters in `Program arguments` (I wrote: `A few input-parameters`)
- Press `Apply` and `Close`

Output

```
Arguments: 3
    A
    few
    input-parameters
```

Java Compilation/Execution



Java in a Terminal Window

Compile/execute `ArgsMain` in package `lecture2`

- The fully qualified name of the program is

  `lecture2.ArgsMain`

- Open a terminal window and go to (using `cd`) the directory containing `lecture2`. For example:

  `C:\software\dv507\src`

- Compile

  ```
  prompt> javac lecture2\ArgsMain.java
  ```

  ⇒ An executable file `ArgsMain.class` is created in the directory `lecture2`

- Execute

  ```
  prompt> java lecture2.ArgsMain  param1 param2 param3
  ```

  ⇒ run `ArgsMain` with the parameters `param1, param2, param3`

Note: This will only work if you have a *complete* java installation!

1. You have downloaded and installed a *Java Development Kit* (jdk)
2. You have configured the environment variables `Path` and `CLASSPATH`

Java Installation (For Windows)

- *Java Runtime Environment* (JRE) (e.g. `jre1.6.0_02`)
  - Everything needed to *execute* Java programs
  - A *Java Virtual Machine* (JVM) (⇒ the command `java`)
    ⇒ `C:\ ... \jre1.6.0_02\bin\java.exe`
  - The Java class library in bytecode format
    ⇒ `C:\ ... \jre1.6.0_02\lib\rt.jar`
  - Present in most operating systems.
- *Java Development Kit* (JDK) (e.g. `jdk1.6.0_04`)
  - Everything needed to *develop* Java programs. JRE (as above) plus...
  - A java compiler (⇒ the command `javac`)
    ⇒ `C:\ ... \jdk1.6.0_04\bin\javac.exe`
  - A JavaDoc generator (⇒ the command `javadoc`)
    ⇒ `C:\ ... \jdk1.6.0_04\bin\javadoc.exe`
- Java API Documentation
  ⇒ HTML documentation of Java's class library.
- Java Source Code
  ⇒ Source code (`.java` filer) to Java's class library.

All are available on Oracle/Java's homepage. Current version: `jdk1.11, update 2`

Note: We are always referring to Java SE (Standard Edition)

Environment variables `Path` and `CLASSPATH`

```
prompt> javac lecture2\ArgsMain.java
'javac' is not recognized as a command, operable program or batch file.
```

- Problem: Windows cannot find the `javac` program.
- `Path` decides where Windows should look for programs.
- We need to add `C:\ ... \jdk1.6.0_04\bin` to the `Path` variable
- Open `Control Panel` ⇒ `System` ⇒ `Advanced` ⇒ `Environmental variables`
- and add `C:\ ... \jdk1.6.0_04\bin` for the variable `Path`

```
prompt>javac lecture2\MP3Main.java
.\lecture2\MP3Track.java:12: package javazoom.jl.player does not exist
.\lecture2\MP3Track.java:20: cannot find symbol: class Player
```

- Problem: Java compiler cannot find the package `javazoom` and the class `Player`
- `CLASSPATH` decides where (`javac/java`) should look.
- Open `Environmental variables` and add the missing path.
  (You might need to create the `CLASSPATH` variable.)
- A minimum is: `CLASSPATH  .;C:\ ... \jre1.6.0_02\lib\rt.jar`
  That is, the present directory and Java's class library.

`prompt> set` ⇒ current environment variables are shown

However, Eclipse takes care of everything!

- Install Eclipse $\Rightarrow$ Eclipse takes care of everything!
- Advantage: Easy to get started using Java
- Disadvantage: You never realise what is going on in the background

If time permits

Try at home to compile and run a Java program using the Terminal Window. It is strongly recommended if you want to learn how Java works.

MP3 in Java programs (External libraries)

Purpose: Learn how to use an external package.

- Looked around on the Internet for MP3 in Java programs.
- We choose `javazoom.JLayer` from `http://www.javazoom.net/projects.html`
  - Also possible to use Java Media Framework and . . . (many available)
- Contents:
  - 10-20 classes (in bytecode) packed in one `.jar`-fil (`javazoom.jar`)
  - Supports the development of programs playing MP3 files
  - Most important: MP3-to-Analogue decoder.
  - Cons: Playing is not threaded $\Rightarrow$ blocks the program.
- Installation in Eclipse:
  1. Download `javazoom.jar` and save it (as for example `C:\software\jars\javazoom.jar`)
  2. Update the Eclipse project Build Path to point the External Archieve `C:\software\jars\javazoom.jar`
  3. Try to execute my test program MP3Main (next slide).

The `.jar` file `javazoom.jar` is a part of the Java Examples for this lecture.

Example Program MP3Main.java

```
 try {
    //String mp3Path = "C:\\software\\java_kurser\\mp3\\";  // My PC
    String mp3Path = "/Users/jlnmsi/Software/java_kurser/mp3/";  // My Mac
    String filename = mp3Path+"Kylie Minogue.mp3";

    FileInputStream fis     = new FileInputStream(filename);
    BufferedInputStream bis = new BufferedInputStream(fis);
    Player player = new Player(bis);
    player.play();                 // Blocks main thread
    System.out.println("Done - MP3 track completed!");
}
catch(JavaLayerException e){
    e.printStackTrace();
}
```

Notice

- The `Player` class comes from `javazoom`
- The print statement is not printed before track completed
  $\Rightarrow$ main thread is blocked!

Exercise 6: XChart

- XChart is a chart library ⇒ xy-plots, pie and bar charts
- XChart website: http://knowm.org/open-source/xchart/

This is an exercise in downloading, installing and using a number of unknown Java packages on the Internet. Therefore, we don't give too much instructions.

Usage Test your installation using ScatterPlot.java (part of lecture examples)

```
public static void main(String[] args) {
    // Create and Customize Chart
    XYChart chart = new XYChartBuilder().width(800).height(600).build();
    chart.getStyler().setDefaultSeriesRenderStyle(XYSeriesRenderStyle.Scatter);
    chart.getStyler().setChartTitleVisible(false);
    chart.getStyler().setLegendPosition(LegendPosition.InsideSW);
    chart.getStyler().setMarkerSize(5);

    // Generate data
    List<Double> xData = new ArrayList<Double>();
    List<Double> yData = new ArrayList<Double>();
    Random random = new Random();
    int size = 1000;
    for (int i = 0; i < size; i++) {
      xData.add(random.nextGaussian() );
```