CE/CS/SE 3354
Software Engineering

Coding Styles

This Class

- Coding styles
  - Code
    - Variables and constants
    - Expressions
    - Statements and Lines
    - Blocks
    - Methods and Classes
    - Enforcing coding styles
  - Comments
    - JavaDoc

Coding Style

- Why?
  - Easier to read: for others and yourself
  - Less mistakes and misunderstandings
  - Reduce the requirement for comments and documentation (self-documented)

Coding Style

- The current status of coding styles
  - Coding style is not a very well organized topic
  - A lot of scattered and conflicting tips by developers

- We organize all these tips from the smallest to the largest code elements (Identifier to File)

Coding Style and Programming Languages

- Coding style is mostly general
  - E.g., should not use too simple variable names
- Coding style can be specific to programming languages
  - Case sensitive / insensitive languages

    Case sensitive: C family, Java, and python

    Case insensitive: Basic, Fortran, Pascal, Ada
  - Indentation based compilation

    Python, Haskell
  - Different code structures

    Classes in OO languages, recursions in functional languages

Variables

- Variable (function, method) naming
  - Do not use names with no meanings (e.g., a, b, ac, xyz)
  - Do not use names with general meanings (e.g., do, run, it, value, temp)
  - Do not differentiate variables with simple number indexes (e.g., name_1, name_2, name_3)
  - Full words is better than abbreviations (e.g., use phone_number instead of tel)
  - Use underscore or Capitalized letter to separate words (e.g., phone_number or phoneNumber)

Variables

- Hungarian Notations
  - Each variable has a prefix to indicate its type

    lAccountNum (long)

    arru8NumberList (array, unsigned 8 bit)
  - Quite useful in C, and also used widely in C++ and C#
  - MS code and Windows APIs use Hungarian Notations a lot
  - Not very necessary in OO languages with modern IDEs, because the type of a variable is complex in itself, and easy to know

Constants

- Constants
  - Give a name to constants

    People know what it is

    Consistent modification

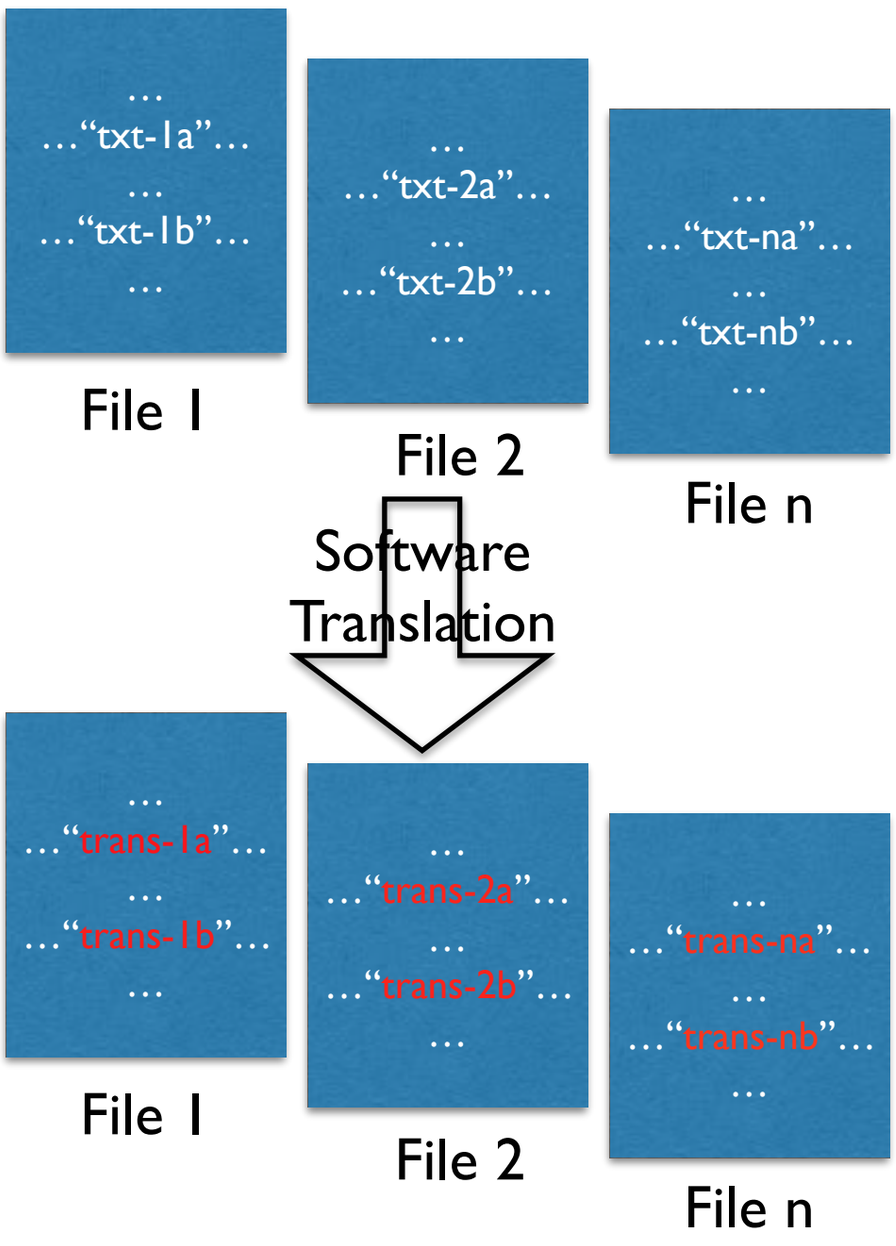
    If global, collect all constants somewhere (e.g, Config class, or a configuration file)

    Conventionally, the name for a class/file-wide constant is all Capitalized words, concatenated by underscore (e.g., MAX_DAY_LIMIT)

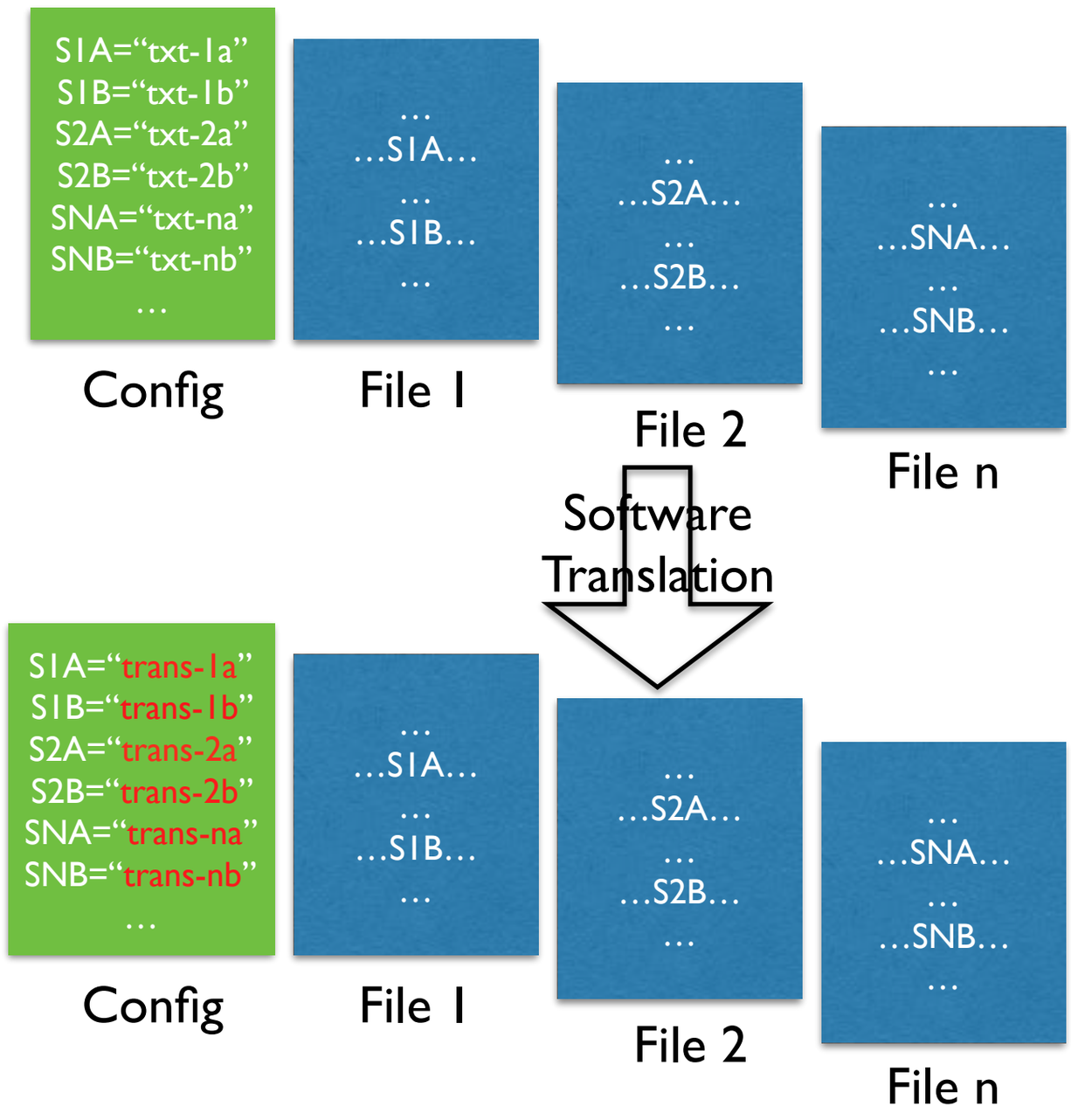
Constants

- String constants
  - Try to avoid string constants in your code
  - Especially if they are related to UI (visible strings), system (paths), database (sql statements), or any environments
  - Put them in different configuration files according to their usage
  - So that you can change them later to generate different releases (different OS, regions, etc.)

String Constants: Example



Bad Style



Good Style

Expressions

- Avoid Complex expressions
  - Give names to intermediate results
  - Example:

```
int totalAmount = (hasDiscount() && validForDiscount(user)) ?
discountRate() * getPrice() * getNum() : getPrice() * getNum();

-------------------------------------------------------------

boolean validDiscount = hasDiscount() && validForDiscount(user);
int price = validDiscount ? getPrice() * discountRate() : getPrice();
int totalAmount = price * getNum();
```

Statements/Lines

- One statement per line
- Line breaks
  - Set a maximum length for lines
  - Break lines at binary operators, commas, etc.
  - Put binary operator at the beginning of the next line.
  - Indent the following sub-lines to differentiate with other code

```
if(long condition 1
        && long condition 2){
    do something;
}
```

Blocks

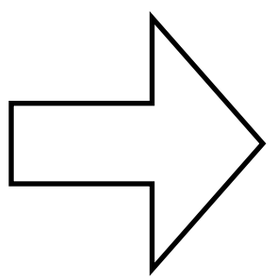
- Indentation
  - Do indentation for blocks (in method, loop, if-else conditions)
  - Pay more intention to indentation in Python and Haskell, it is a right or wrong thing!!
  - Try to avoid using tab, why?

    Tabs are translated to 2 or 4 spaces in different editors or settings, resulting in global coding style problems, and the case is even worse for Python
  - 2 (scala, ruby, …) or 4 spaces (others), usually depending on the language you use and the team convention you decided on

Blocks

- Single statement blocks for loop and conditions
  - Add curly brackets around, why?

```
if(a > 1)
    do something;
another statement;
```



```
if(a > 1)
//    do something;
another statement;
```

- Logic blocks
  - Divide a basic block to several logic blocks, add an empty line between them

```
int validDiscountRate = …;
int finalAmount = validDiscountRate * amount;


print getReceiptHeader();
print finalAmount;
```

Blocks

- Code Clones
  - Copy and paste is a notorious programming habit
  - Simple but result in code clones
    - Inconsistent modification of different clone segments
- Example:

```
for (int i = 0; i < 4; i++){
   sum1 += array1[i];
}
average1 = sum1/4;

for (int i = 0; i < 4; i++){
   sum2 += array2[i];
}
average2 = sum1/4;
```

Blocks

- Code Clone Detection
  - Quite a number of tools

    CCFinder, Deckard, Conqat, mostly for C, C++ and Java. Visual Studio also has the feature for C#
- After Detection
  - Eliminate (extract a method invoked by both places)
  - Sometimes not easy

    Change other people's code

    20% code clones in industry / open source code
  - Try to keep track of them

Methods

- Size
  - Smaller, and smaller, and even smaller …
  - Usually less than 20 lines, and never exceed 100 lines
  - Why?

    Easier to understand

    Easier to reuse: less code clones

Methods

- Signatures
  - Avoid too many parameters
  - May consider group parameters to a new class
- Contents
  - Try to avoid side effect
  - Try to not use global variables, especially writing global variables

File Structure

- Follows the common structure of the language (e.g., imports, class decl, fields, methods in Java, includes, macros, structs, functions in C)
- Keep it small
  - Usually no more than 500 lines
  - Should break a module to multiple sub-modules (it is also a design issue)
- Put public methods before private methods
  - Easier for other people to read public methods

This Class

- Coding styles
  - Code

    Variables and constants

    Expressions

    Statements and Lines

    Blocks

    Methods and Classes

    Enforcing coding styles
  - Comments
  - JavaDoc

Comments

- A way to help other people and yourself to better understand your code later
- Common rules
  - No need to comment everything
  - More important to explain variables than statements: Other developers know what the statement does, but do not know what the variable means…
  - Try to write complete sentences
  - Avoid code in comments
  - Clean commented code ASAP

Comments for Statements

- Two types of statements especially require comments
  - Declaration statements
  - Statements with complex expressions
- No meaningless comments
  - Example:

```
int score = 100; // set score to 100
```

```
int score = 100; // the full score of the final exam is 100
```

Comments for Methods

- A well-structured format for each public method
- Explain:
  - What the method does
  - What does the return value mean
  - What does the parameters represent
  - What are restrictions on parameters
  - What are the exceptions, and what input will result in exceptions
- For private method, comment may be less structured, but still need to include the information

Comments for Methods

- Example:

```
/**
 * Fetch a sub-array from an item array. The range
 * is specified by the index of the first item to
 * fetch, and ranges to the last item in the array.
 *
 * @param list represents the item list to fetch from
 *                  it should not be null.
 * @param start represents the start index of the
 *                   fetching range it should be between
 *                   0 and the length of list
 * @return the fetched subarray
 */
public List<Item> getRange(List<Item> list, int start){
```

Comments for Classes/Files

- A well-structured comment for each class / file
- Include:
  - Author (s)
  - Generating time and version
  - Version of the file
  - A description of the class/file about its main features, and usage specifications / examples

Comments for Classes/Files

- Example:

```
/**
 * A mutable sequence of characters. The principal operations on a StringBuilder
 * are the append and insert methods. The append method always adds these
 * characters at the end of the builder; the insert method adds the characters at
 * a specified point.
 *
 * For example, if z refers to a string builder object whose current contents are
 * "start", then the method call z.append("le") would cause the string builder to
 * contain "startle", whereas z.insert(4, "le") would alter the string builder to
 * contain "starlet".
 *
 * author    :   John Smith
 * generate:   Oct.1st.2013
 * version   :  1.0
*/
public class StringBuilder{
```

Code Convention for Companies / Organizations

- Most companies / organizations have their code conventions
  - Usually following the general rules of coding style
  - May set different line-length limits, 2 or 4 spaces, different ways to place braces, or whether to put spaces around operators or brackets, …
  - Some code convention examples:

    Java conventions: http://www.oracle.com/technetwork/java/codeconvtoc-136057.html

    C# conventions: http://msdn.microsoft.com/en-us/library/vstudio/ff926074.aspx

Coding Style Enforcement: Eclipse

- Set up formatter for code
  - Go to Eclipse -> preferences -> Java (or other languages installed) -> coding styles -> formatter
  - Select an existing profile (e.g. eclipse built-in) or click on New… to generate a new one
  - Click on edit to open the edit panel
  - Edit the profile: change brace rules, tab to space, etc.
  - Right click on your project -> properties -> Java coding styles -> formatter-> check project specific -> choose a formatter and click apply

Today's Class

- Coding styles
  - Code
    - Variables and constants
    - Expressions
    - Statements and Lines
    - Blocks
    - Methods and Classes
    - Enforcing coding styles
  - Comments
    - JavaDoc

JavaDoc

- A JavaDoc Example
  - Ant 1.6.5
  - http://api.dpml.net/ant/1.6.5/overview-summary.html

JavaDoc

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS
SUMMARY: NESTED | FIELD | CONSTR | METHOD

org.apache.tools.ant.filters.util

Class ChainReaderHelper

```
java.lang.Object
  |
  +--org.apache.tools.ant.filters.util.ChainReaderHelper
```

public final class **ChainReaderHelper**
extends java.lang.Object

Process a FilterReader chain.

Field Summary

int	**bufferSize** The size of the buffer to be used.
java.util.Vector	**filterChains** Chain of filters
java.io.Reader	**primaryReader** The primary reader to which the reader chain is to be attached.

Constructor Summary

ChainReaderHelper()

Method Summary

java.io.Reader	**getAssembledReader**() Assemble the reader

- A Ja
  - A
  - h html

JavaDoc-Tags

- Tags are meta-data put in the comments to guide Javadoc in document generation
- Starting with "@"

  @author : author of the class

  @version: current version of the software

  @since: the software version when the class is added

  @see: a link to another class / method / …

  @param: parameters of a method

  @return: return value of a method

  @exception: exceptions thrown from a method

Javadoc: Mapping

- Commenting methods for documentation

```
/**
 * Constructs a WordTree node, specify its
 * data with the parameter word, both children
 * of the WordTree node are null
 *
 * @param word the string value stored as the data of the node, cannot be null
 * @exception NullWordException an exception raised when the parameter word is null
 */
public WordTree(String word) throws NullWordException{
...
/**
 * fetch the data of current WordTree node
 * @return the data stored at current node
 */
public String getData()
...
```

Javadoc: Mapping

- Javadoc generated

Constructor Detail

WordTree

```
public WordTree(java.lang.String word)
         throws NullWordException
```

Constructs a WordTree node, specify its data with the parameter word, both children of the WordTree node are null

Parameters:

`word` the string value stored as the data of the node, cannot be null

Throws:

`NullWordException` - an exception raised when the parameter word is null

Method Detail

getData

```
public java.lang.String getData()
```

fetch the data of current WordTree node

Returns:

the data stored at current node

JavaDoc in Eclipse

- Generating JavaDoc in eclipse
  - Right click on your project/package/class/…
  - Goto Export…, and choose JavaDoc
  - You will need a Java SDK (not only JRE) to run JavaDoc
  - Set the destination to the place
  - Click on finish