Unit 11.3B: System programming

Good programming style

Learning objectives :

11.5.3.12 use good programming style rules when writing code

```
/**
* Code Readability
*/
if (readable()) {
    be_happy();
} else {
    refactor();
}
```

Using a disciplined style of programming will save you time whenever you have to read your program or debug it.

1. What is a good programming style?
2. In order to maintain a good programming style, what points should be taken into account?
3. What rules of good code can you provide?
4. Why is it important to follow the rules of good code?

Some more specific advice on good coding style

Code formatting:

•Indent your code so that it corresponds to the logical structure of the program.

(If you edit your code in a Java programming environment, the pretty printer will already take of the layout.)

Which code is good *example* and which is bad *example*?

```
for(int i=0; ... )
for(int j=0; ... )
if(i<j)
...
```

```
for(int i=0; ... )
    for(int j=0; ... )
        if(i<j)
            ...
```

Some more specific advice on good coding style

Code formatting:

•Indent your code so that it corresponds to the logical structure of the program.

(If you edit your code in a Java programming environment, the pretty printer will already take of the layout.)

Which code is good *example* and which is bad *example*?

```
for(int i=0; ... )
for(int j=0; ... )
if(i<j)
...
```

Bad example:

```
for(int i=0; ... )
    for(int j=0; ... )
        if(i<j)
            ...
```

Good example:

•Try to create a good measure of vertical density of your code (new lines between commands) and keep it consistent

Which code is good *example* and which is bad *example*?

```
public static int MethodX()
{   int pointX; int pointY; int pointZ;
    ...
    return pointX+pointY+pointZ; }
public static int MethodY()
{  ... }
```

```
public static int MethodX(){
    int pointX; // comment ...
    int pointY; // comment ...
    int pointZ; // comment ....

    ...

    return pointX+pointY+pointZ;
}


public static void MethodY(){
    ...
}
```

•Try to create a good measure of vertical density of your code (new lines between commands) and keep it consistent

Which code is good *example* and which is bad *example*?

Good example:

```
public static int MethodX(){
    int pointX; // comment ...
    int pointY; // comment ...
    int pointZ; // comment ....

    ...

    return pointX+pointY+pointZ;
}


public static void MethodY(){
    ...
}
```

```
public static int MethodX()
{   int pointX; int pointY; int pointZ;
    ...
    return pointX+pointY+pointZ; }
public static int MethodY()
{  ... }
```

Bad example:

Variables:

•Use meaningful variable names. The name should reveal the usage intent.

Which code is good *example* and which is bad *example*?

```
int currentMax;
```

Good example:

```
int cMX;
```

Bad example:

•If the variable name does not sufficiently describe the usage intent, comment on the variable at its declaration.

•It is a good practice to declare a global variable for each parameter of the method that is given as a concrete value.

Which code is good *example* and which is bad *example*?

```
public static int FIRST_ITERATION = 15;
public static int SECOND_ITERATION = 7;

public static void MethodX(){
    ...
     for (... ;i<FIRST_ITERATION; ...)
        for (... ;j<SECOND_ITERATION; ...)
    ...
}
```

```
public static void MethodX(){
    ...
     for (... ;i<15; ...)
        for (... ;j<7; ...)
    ...
}
```

•It is a good practice to declare a global variable for each parameter of the method that is given as a concrete value.

Which code is good *example* and which is bad *example*?

```
public static int FIRST_ITERATION = 15;
public static int SECOND_ITERATION = 7;

public static void MethodX(){
    ...
     for (... ;i<FIRST_ITERATION; ...)
        for (... ;j<SECOND_ITERATION; ...)
    ...
}
```

Good example:

```
public static void MethodX(){
    ...
     for (... ;i<15; ...)
        for (... ;j<7; ...)
    ...
}
```

Bad example:

Testing:

- Every method written should be tested (even the most trivial one).
- The test should be done in a way that exemplifies the use of code.
- We recommend that for each developed class one develops a corresponding testing class where each method is tested using one or more methods.
- Be suspicious when testing – test for expected outputs but also for unexpected outputs.

Good example:

```
public static void testFindInSortedArray(){
    int [] array = { -11, -3, -1, 2, 8, 11, 15, 23};

    // Here we expected that method finds the element with index 6
    System.out.println(findInSortedArray(array,15));

    // Here the method should return not found (-1)
    System.out.println(findInSortedArray(array,3));
}
```

Commenting Code:

- Writing comments is an integral part of the coding – use both multi-line comments (e.g., /* */) and in-line comments (e.g., //) where needed.
- Do not over-comment your code – find a good balance.

Bad example:

```
/* this is the main method where our program starts*/
public static void main(String[] args) {
    ...
}
```

Bad example:

```
// methodTryThis(3);
// methodTryThat(4);
// methodTryThis(5);
// methodTryThat(6);
```

- Avoid leaving code lines that are commented out in the final version of your code.

http://www.inf.unibz.it/~nutt/Teaching/DSA1415/DSAAssignments/good-coding-style.html

6 Best Rules for Good Programming Style

Readability
Maintainability
Comments
Naming
Indentation
Output


```
/**
* Code Readability
*/
if (readable()) {
    be_happy();
} else {
    refactor();
}
```

Successful criteria: can explain the Rules for Good Programming Style

Readability

Good code is written to be easily understood by colleagues. It is properly and consistently formatted and uses clear, meaningful names for functions and variables. Concise and accurate comments describe a natural decomposition of the software's functionality into simple and specific functions. Any tricky sections are clearly noted. It should be easy to see why the program will work and reason that it should work in all conceivable cases.

Maintainability

Code should be written so that it is straightforward for another programmer to fix bugs or make changes to its functionality later. Function should be general and assume as little as possible about preconditions. All important values should be marked as constants which can be easily changed. Code should be robust to handle any possible input and produce a reasonable result without crashing. Clear messages should be output for input which is not allowed.

Comments

Comments are the first step towards making computer program human readable. Comments should explain clearly everything about a program which is not obvious to a peer programmer. The volume of comments written is meaningless, quality is all that counts. They should go at the top of every source
file and generally include your name, the date your code was written and overall description of the purpose of that program.

Naming

Names given to classes, variables, and functions should be unambiguous and descriptive. Other guidelines for naming are:

- Capitalization is used to separate multi-word names: StoneMasonKarel.
- The first letter of a class name is always capitalized: GraphicsProgram
- The first letter of a function or variable name is always in lowercase: setFilled().
- The names x and y should only be used to describe coordinates.
- The names i, j, and k should only be used as variables in for loops.

Indentation

Indentation is used to clearly mark control flow in a program. Within any bracketed block, all code is indented in one tab. This includes the class body itself. Each additional for, while, if, or switch structure introduces a new block which is indented, even if brackets are omitted for one line statements. For if statements, any corresponding else statements should line up

Output

A final, overlooked aspect of good programming style is how our program output results and information to users. Part of writing professional looking programs is providing clear instructions and results to the users of our programs. This means proper English with no spelling error conditions. One must always assume that writing programs to be used by somebody with no understanding of computer programming whatsoever. If you liked the article then please share it!

Readability

Good code is written to be easily understood by colleagues. It is properly and consistently formatted and uses clear, meaningful names for functions and variables. Concise and accurate comments describe a natural decomposition of the software's functionality into simple and specific functions. Any tricky sections are clearly noted. It should be easy to see why the program will work and reason that it should work in all conceivable cases.

Maintainability

Code should be written so that it is straightforward for another programmer to fix bugs or make changes to its functionality later. Function should be general and assume as little as possible about preconditions. All important values should be marked as constants which can be easily changed. Code should be robust to handle any possible input and produce a reasonable result without crashing. Clear messages should be output for input which is not allowed.

Comments

Comments are the first step towards making computer program human readable. Comments should explain clearly everything about a program which is not obvious to a peer programmer. The volume of comments written is meaningless, quality is all that counts. They should go at the top of every source
file and generally include your name, the date your code was written and overall description of the purpose of that program.

Naming

Names given to classes, variables, and functions should be unambiguous and descriptive. Other guidelines for naming are:

- Capitalization is used to separate multi-word names: StoneMasonKarel.
- The first letter of a class name is always capitalized: GraphicsProgram
- The first letter of a function or variable name is always in lowercase: setFilled().
- The names x and y should only be used to describe coordinates.
- The names i, j, and k should only be used as variables in for loops.

Indentation

Indentation is used to clearly mark control flow in a program. Within any bracketed block, all code is indented in one tab. This includes the class body itself. Each additional for, while, if, or switch structure introduces a new block which is indented, even if brackets are omitted for one line statements. For if statements, any corresponding else statements should line up

Output

A final, overlooked aspect of good programming style is how our program output results and information to users. Part of writing professional looking programs is providing clear instructions and results to the users of our programs. This means proper English with no spelling error conditions. One must always assume that writing programs to be used by somebody with no understanding of computer programming whatsoever. If you liked the article then please share it!

Summary

Code should be easy to read.

It should be easy to understand what the programmer means.

It should be easy to see how the different parts of the program fit into the whole.

Don’t duplicate code.

Write modular code. Different concerns should be handled by different classes, and where possible even different methods within one class.

Formative assessment `Programming Style`

11.5.3.12 use good programming style rules when writing code

This example shows how a thrown exception is passed back to the calling method, and to the method that called that one, and so on until a catch clause is found or the main method is reached.

```java
public class Example {

    public static void main(String[] args) {

        try {
            Example example = new Example();
            int amount = example.getAccountBalanceByName("John Mcleod");
        }
        catch (UnknownCustomerException e) {
            System.out.println("Couldn't find id for customer");
        }
    }
    /**
     * Gets the account balance for the customer with the given name.
     *
     * @param customerName the name of the customer
     * @return the current account balance of the customer
     * @throws UnknownCustomerException if a customer with the given name can't be found
     */
    public int getAccountBalanceByName(String customerName) throws UnknownCustomerException {

        int id = getId(customerName);
        return getAccountBalanceById(id);
    }

    private int getId(String customerName) throws UnknownCustomerException {

        // In a real implementation this would look up the name in an array, database etc.
        if (customerName.equals("Jane Smith")) {
            return 32576;
        }
        else {
            throw new UnknownCustomerException();
        }
    }

    private int getAccountBalanceById(int customerId) throws UnknownCustomerException {

        // In a real implementation this would look up the balance in an array, database etc.
        return -10;
    }
}
```

Here is a class written in a common Java style, illustrating a number of the style guidelines.
Try to do good style if you prefer a different style is considered acceptable and explain.

- *Compare values*
- *Simplicity*
- *Clarity of nested comparisons*
- *Unnecessary variables*

For example:

This example illustrates:

- *single spaces between most reserved words, identifiers and symbols*
- *single blank lines between methods*
- *……..*
- *……..*

How do I feel about the lesson?

5	I feel very confused or frustrated. I need help.	
4	I feel confused. I have questions.	
3	I feel so-so. I am starting to get it.	
2	I get it. I am feeling good.	
1	I get it completely. I feel like an expert!	

Programming

Using Trace tables and Good Programming Techniques

Objectives

At the end of the lesson students should be able to

✓State the purpose of writing good codes

✓Describe the different programming techniques

✓Use trace table to dry run an algorithm

GOOD PROGRAMMING TECHNIQUES

PROGRAMMING LANGUAGES

Algorithms are designed to solve problems whereby any code written in a programming languages is often used to implement algorithms.

They create the programs (**software**) that communicate instructions to a computer.

All programming languages have the ability to:

- **input** data from a device such as a keyboard
- **output** data to a device such as a screen
- **process** input data such as **calculations, make decisions** based on certain conditions being met, make **repetition** for a certain number of times, or while a condition is met, or until a condition is met

Good Programming Style

- Programming style is a term used to describe the effort a programmer should take to make his or her code easy to read and easy to understand.
- Good organization of the code and meaningful variable names help readability, and the use of comments can help the reader understand what the program does and why.
- Proper programming style significantly reduces maintenance
- costs and increases the lifetime and functionality of software. Most software
- disasters are rooted in poor style of programming

Indentation

The purpose of code indentation and style

- is to make the program more readable and understandable.
- It saves lots of time while we revisit the code and use it

Lines are indented by four spaces to indicate that they are contained within a statement in a previous line.

```
IF average ≥ 50 THEN
        OUTPUT "Pass"      This is indented as it is dependent
                           on the 'IF' statement.
ELSE
        OUTPUT "Fail"      This is indented as it is dependent
                           on the 'ELSE' statement.
ENDIF
```

Meaningful Variable Names

Another issue to a program readability is that the program's identifiers (variable names, subprogram names, etc.) are mostly meaningless.

You should strive to give each object a name that gives the reader a strong hint as to the object's purpose within the program.

Many early languages limited the size of the allowable names, and that forced programmers to use short, meanful names.

Modern languages permit identifier names to be quite lengthy, so there's no excuse not to create good names

Principles in Naming Variables

- Use good, meaningful names, If you have a variable in your program that holds the number of hours an employee works in a week, you might call it HOURS or HOURS_PER_WEEK is a good solution,
- Use underscores as part of names. If you can't use them, you can still improve name readability by mixing the case of the letters in the name. example 'HoursPerWeek' is much easier to read than 'hoursperweek'.
- After each variable declaration place a comment that explains the purpose of the variable.
- Common abbreviations are often acceptable in variable names. For example: HRS_PER_WEEK.

Consistent Naming Scheme

The names given to variables, constants, procedures and functions should be descriptive as possible. Using descriptive names improves the overall quality of the software so it makes it much easier to modify and read the code and allows us to easily understand what the variable, procedure or function does or what it is used for.

Identifiers/names should have word boundaries.

There are two popular options:

- **camelCase:** First letter of each word is capitalized, except the first word.
- **underscores:** Underscores between words, like: mysql_real_escape_string().

COMMENTS

A comment is one or more sentences that explain the purpose of a section of **code**.

A comment is placed just before (or after) the code to which it refers

Well-written comments explain:

- the name of the **program** and its author
- what the programmer intends the code to do
- whether the programmer thinks the code could be better written
- where the program may be incomplete or need updating
- Comments make your code easier to understand

Different programming uses different symbol to indicate a comment

In **Python**, comments start with a hash symbol – '#'.

Comments are preceded by two forward slashes //

```
# short program to show how variable values can change
brush_width ← 242
draw_square(brush_width)

brush_width ← 187
draw_square(brush_width)

brush_width ← brush_width - 20
draw_square(brush_width)
```

```
IF average ≥ 50 THEN
        OUTPUT "Pass"      This is indented as it is dependent
                           on the 'IF' statement.
ELSE
        OUTPUT "Fail"      This is indented as it is dependent
                           on the 'ELSE' statement.
ENDIF
```

Whitespaces

- These empty spaces and lines are called **whitespace**.
- Adding whitespace breaks up your code visually and makes it much easier to read
- Reader can see at a glance the statement for each block

```php
<?php
include ('connrssnormal.php');
echo f();
$c->disconnect();
function f() {
global $c;
$query = "SELECT * FROM webref_rss_details";
$result = $c->query($query);
if ( mysql_affected_rows() > 0 ){
$a = $result->numRows();
} else {
$a = 0;
}
for ($i=0; $i < $a; $i++) {
$row = $result->fetchRow(DB_FETCHMODE_ASSOC);
$details = '<rss version="2.0">
<channel>
<title>'. $row['title'] .'</title>
<link>'. $row['link'] .'</link>
<description>'. $row['description'] .'</description>
<language>'. $row['language'] .'</language>
<image>
<title>'. $row['image_title'] .'</title>
<url>'. $row['image_url'] .'</url>
<link>'. $row['image_link'] .'</link>
<width>'. $row['image_width'] .'</width>
<height>'. $row['image_height'] .'</height>
</image>';
}
return $details;
}
```

Code written in PHP

The programmer has given no explanations or structure, making it very difficult to follow.

Perfectly working code!

Which means you can write functional code, but it takes skill and professionalism to make it maintainable.

```php
<?php
include ('connrssnormal.php');
echo f();
$c->disconnect();
function f() {
global $c;
$query = "SELECT * FROM webref_rss_details";
$result = $c->query($query);
if ( mysql_affected_rows() > 0 ){
$a = $result->numRows();
} else {
$a = 0;
}
for ($i=0; $i < $a; $i++) {
$row = $result->fetchRow(DB_FETCHMODE_ASSOC);
$details = '<rss version="2.0">
<channel>
<title>'. $row['title'] .'</title>
<link>'. $row['link'] .'</link>
<description>'. $row['description'] .'</description>
<language>'. $row['language'] .'</language>
<image>
<title>'. $row['image_title'] .'</title>
<url>'. $row['image_url'] .'</url>
<link>'. $row['image_link'] .'</link>
<width>'. $row['image_width'] .'</width>
<height>'. $row['image_height'] .'</height>
</image>';
}
return $details;
}
```

```php
// Written by:  John Brown
// Date: 18th Dec 2016
// Version: 2.3
// ******************************************************
include ('connrssnormal.php');    // connection function for the database
echo gettRSSdetails();  // print out the details to the screen
$db->disconnect();  // Disconnect from the database now that all records are displayed

//*******************************************************
//*  This will access the RSS database and extract
//*  every record from it. Then it formats each record as XML and returns
//*  the complete set to the calling code
//*  the getItems() function needs to be called after this to close off the xml properly
function getRSSdetails() {
    global $db; // $db has been initialised by the connrssnormal function above
    $query = "SELECT * FROM webref_rss_details";  // Get all records from the database
    $result = $db->query($query);  // Store the results as an associative array

    if ( mysql_affected_rows() > 0 ){    // deal with no records being returned
        $num_results = $result->numRows();
        } else {
        $num_results = 0;
    }

    for ($i=0; $i < $num_results; $i++) {     // loop over the results array
        $row = $result->fetchRow(DB_FETCHMODE_ASSOC);  // handle one record at a time

        // Add the XML formatting code to each record entry
        $details = '<rss version="2.0">
        <channel>
        <title>'. $row['title'] .'</title>
        <link>'. $row['link'] .'</link>
        <description>'. $row['description'] .'</description>
        <language>'. $row['language'] .'</language>
        <image>
        <title>'. $row['image_title'] .'</title>
        <url>'. $row['image_url'] .'</url>
        <link>'. $row['image_link'] .'</link>
        <width>'. $row['image_width'] .'</width>
        <height>'. $row['image_height'] .'</height>
        </image>';
    }

return $details;  // XML formatted record returned
}
```