**LAB # 2**

**Good practices of programming**

**OBJECTIVE**

Implementing good code practices and code optimization techniques.

**Java Naming conventions**

Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method, etc.

But, it is not forced to follow. So, it is known as convention not rule. These conventions are suggested by several Java communities such as Sun Microsystems and Netscape.

All the classes, interfaces, packages, methods and fields of Java programming language are given according to the Java naming convention. If you fail to follow these conventions, it may generate confusion or erroneous code.

**Advantage of naming conventions in java**

By using standard Java naming conventions, you make your code easier to read for yourself and other programmers. Readability of Java programs is very important. It indicates that less time is spent to figure out what the code does.

The following are the key rules that must be followed by every identifier:

* The name must not contain any white spaces.
* The name should not start with special characters like & (ampersand), $ (dollar), \_ (underscore).

Let's see some other rules that should be followed by identifiers.

1. **Class**

Classes and objects should have noun or noun phrase names like Customer, WikiPage, Account, and AddressParser. Avoid words like Manager, Processor, Data, or Info in the name of a class. A class name should not be a verb.

* It should start with the uppercase letter.
* It should be a noun such as Color, Button, System, Thread, etc.
* Use appropriate words, instead of acronyms.
* **Example: -**

1. **public** **class** Employee
2. {
3. //code snippet
4. }
5. **Interface**

* It should start with the uppercase letter.
* It should be an adjective such as Runnable, Remote, Action Listener.
* Use appropriate words, instead of acronyms.
* **Example: -**

1. **interface** Printable
2. {
3. //code snippet
4. }
5. **Method**

Methods should have verb or verb phrase names like postPayment, deletePage, or save. Accessors, mutators, and predicates should be named for their value and prefixed with get, set, and is according to the java bean standard.4

string name = employee.getName();

customer.setName("mike");

if (paycheck.isPosted())...

* It should start with lowercase letter.
* It should be a verb such as main(), print(), println().
* If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().
* **Example:-**

1. **class** Employee
2. {
3. //method
4. **void** draw()
5. {
6. //code snippet
7. }
8. }
9. **Variable**

* It should start with a lowercase letter such as id, name.
* It should not start with the special characters like & (ampersand), $ (dollar), \_ (underscore).
* If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName.
* Avoid using one-character variables such as x, y, z.
* **Example: -**

1. **class** Employee
2. {
3. //variable
4. **int** id;
5. //code snippet
6. }
7. **Package**

* It should be a lowercase letter such as java, lang.
* If the name contains multiple words, it should be separated by dots (.) such as java.util, java.lang.
* **Example :-**

1. **package** com.javatpoint; //package
2. **class** Employee
3. {
4. //code snippet
5. }
6. **Constant**

* It should be in uppercase letters such as RED, YELLOW.
* If the name contains multiple words, it should be separated by an underscore(\_) such as MAX\_PRIORITY.
* It may contain digits but not as the first letter.
* **Example :-**

1. **class** Employee
2. {
3. //constant
4. **static** **final** **int** MIN\_AGE = 18;
5. //code snippet
6. }

In the code given below, the class, variables, methods and objects do not have meaningful names. Find non-meaningful names and convert into better naming conventions.

|  |
| --- |
| public class res {  private String firstname;  private String lastname;    public res()  {  }  public String getfn() {  return firstname;  }  public void setfn(String firstname) {  this.firstname = firstname;  }  public String getln() {  return lastName;  }  public void setln(String lastname) {  this.lastname = lastname;  }    public static void main(String args[]) {      }  }  class main{      public static void main(String args[]) {  String sr;  res rs=new res();  rs.setfn("Hiba");  sr=rs.getfn();  System.out.println(sr);    }  } |

**Code Optimization:**

In software development and program implementation mechanisms, there are certain types of techniques that should be designed for perfect execution of the program. One of the best techniques of ensuring program quality is optimization. The code optimization is applied on any method of code to improve code quality and its efficiency. After applying code optimization tricks, the program would be smaller in size, consumes less memory, executed more rapidly and performs fewer input and output operations.

Optimization can be performed by automatic optimizers, or programmers. An optimizer is either a specialized software tool or a built-in unit of a compiler (the so-called optimizing compiler). Modern processors can also optimize the execution order of code instructions.

Optimizations are classified into high-level and low-level optimizations. High-level optimizations are usually performed by the programmer, who handles abstract entities (functions, procedures, classes, etc.) and keeps in mind the general framework of the task to optimize the design of a system. Optimizations performed at the level of elementary structural blocks of source code - loops, branches, etc. - are usually referred to as high-level optimizations too, while some authors classify them into a separate ("middle") level (N. Wirth?). Low-level optimizations are performed at the stage when source code is compiled into a set of machine instructions, and it is at this stage that automated optimization is usually employed. Assembler programmers believe however, that no machine, however perfect, can do this better than a skilled programmer (yet everybody agrees that a poor programmer will do much worse than a computer).

**Advantages:**

 The optimized code has the following advantages-

* Optimized code has faster execution speed.
* Optimized code utilizes the memory efficiently.
* Optimized code gives better performance.

**Code Optimization Techniques:**

Following are the most important techniques of code optimization:

1. **Compile Time Evaluation**

In this technique, such values are evaluated before code compilation. We have a simple example of constant value of circumference of circle i.e. 3.14. But if any programmer writes 22/7 than at compile time execution time will be more required. Therefore, the compile time evaluation provides us a simpler to use optimistic and solved value instead of calculated operations.

Un-optimized value = 22/7

Optimized value = 3.14

1. **Common subexpression elimination**

It is a very common practice of initializing variables for program operations. But sometimes repeated values are initialized in different variables. Therefore, it would be eliminated to avoid re-computation

Un-optimized value:   S1 = 4 x i;

S2 = a[S1]

S3 = 4 x j

S4 = 4 x i **// Redundant Expression**

S5 = n

Optimized value:      S1 = 4 x i

S2 = a[S1]

S3 = 4 x j

S5 = n

1. **Dead Code Elimination**

A piece of code that has no courage to be executed, or its output is unreachable. Hence, this type of dead code is eliminated for improving performance of the system.

Un-optimized code:

i = 0 ;

if (i == 1)

{

a = x + 5 ;

}

Optimized code:

i = 0 ;

1. **Code Movement**

Code movement is a mechanism where the value is separated out from unwanted iteration. It means, the code present inside the loop is moved out if it does not matter whether it is present inside or outside. Such a code unnecessarily gets executed again and again with each iteration of the loop. This leads to the wastage of time at run time.

Un-optimized value:

for (int j = 0 ; j < n ; j ++)

{

x = y + z ;

a[j] = 6 x j;

}

Optimized value:

x = y + z ;

for ( int j = 0 ; j < n ; j ++)

{

a[j] = 6 x j;

}

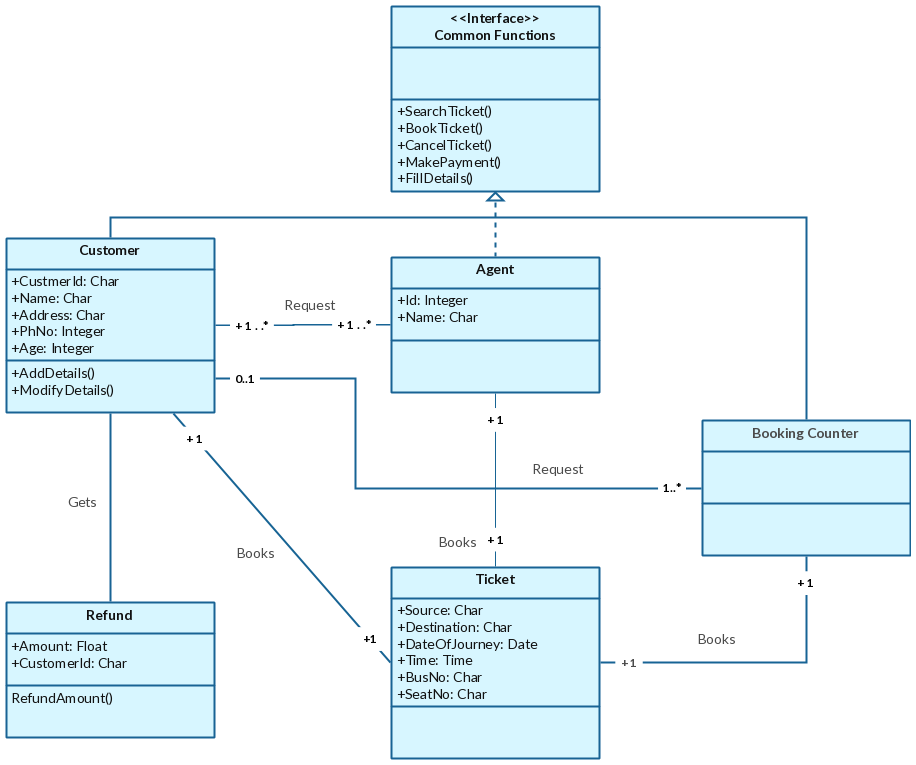
1. **Strength Reduction**

In this technique the expensive and costly operators are reduced. Like the operator of multiplication is more heavily executed than the operator of addition. Therefore, whenever easier calculation could be programmed it will enhance the performance of the code.

Un-optimized code: B = A x 2

Optimized code: B = A + A

**Lab Task:**

1. Create a design for the mark sheet by taking runtime value of student name, total marks, obtained marks and calculate its percentage, grade and GPA. Use good practices of programming that we have studied and ensure that the outcomes should be presented in a proper Viewable approach.
2. Create a class Rectangle with attributes length and width, each of which defaults to 1. Provide methods that calculate the rectangle’s perimeter and area. It has set and get methods for both length and width. The set methods should verify that length and width are each floating-point numbers larger than 0.0 and less than 20.0. Write a program to test class Rectangle.
3. Convert the following class diagram into optimized and meaningful java code.