**SEPQM Assignment 2**

**Introduction of new complexity metric**

**Group Details**

|  |  |  |  |
| --- | --- | --- | --- |
| **Group ID** | **2021S1\_REG\_WE\_02** | | |
| **Name** | | **Registration No.** | **Responsible Factors** |
| Wickramarathna W.G.M.S | | IT19004778 | * Number of inputs and outputs in each class * Nested levels * Threading |
| De Silva K.H.K.L | | IT19006994 | * Number of executable lines of code * Inheritance Levels including interfaces. * Types of Access Modifiers used in Encapsulation. |
| Lekamalage U.L.V.M. | | IT19111766 | * Overloading and Overriding in polymorphism * Number of parameters in a method * Token count |
| Hakeem M.U.M.I | | IT19009964 | * Number of functions in a class * Control structures * Exception Handling |

* Advantages and limitations of the CC metric

|  |
| --- |
| Advantages   1. CC metric can be used as an ease of maintenance metric. 2. Used as a quality metric, gives relative complexity of various designs. 3. It can be computed early in life cycle than other metrics. 4. Measures the minimum effort and best areas of concentration for testing. 5. It guides the testing process by limiting the program logic during development. 6. It easy to apply.   Limitations   1. The CC is a measure of the program’s control complexity and not the data complexity. 2. The same weight is placed on nested and non-nested loops. When deeply nested loops in the code its harder to understand than non-nested structures. 3. It may give a misleading figure regarding a lot of simple comparison and decision structures. 4. This metric only considers a program complex which has a greater number of decision and control statements. It ignores the size of the program or size of the code under each node. |

* Advantages and limitations of the WCC metric

|  |
| --- |
| Advantages   1. Appropriate for object-oriented software development and supports classes, inheritance, and polymorphism. 2. Support coupling and cohesion 3. Characteristics such as program flow control and nesting have considered to develop the metrics. 4. Keep track of features of the software system such as size in terms of operands and operators.   Limitations   1. Methods and the data attributes are responsible for the complexity of a class. But WCC metrics do not consider the internal architecture of methods and the attributes together. 2. Cognitive functionalities express the effort and time required to comprehend a given software. In WCC cognitive characteristics have not been considered for the complexity of the code. 3. WCC does not consider the access modifiers of the methods in encapsulation. 4. WCC metrics are too implementation technology dependent. |

* Advantages and limitations of the of CFS metric

|  |
| --- |
| Advantages   1. Best suitable for small programs. 2. Easy to calculate. 3. Considers the basic control structures (BCS), number of inputs and number of outputs measuring complexity.   Limitations   1. Trying to calculate the complexity of a system can be time-consuming, if the program is very large. 2. Some of the important object-oriented design aspects like inheritance are ignored by this metric. 3. Does not consider operators and identifiers when calculating the complexity. |

* How the limitation of the CC, WCC, and CFS metrics be overcome

|  |
| --- |
| **How the limitations of the CC metric be overcome**  We should avoid CC metric’s limitations such as nested loops problem, misleading figure regarding, and only measure of the program’s control complexity. To overcome those limitations, we should have a way to understand and allocate nested loops value, and provide a way to measure data complexity, not only that CC metrics provide misleading figures regarding a lot of simple comparisons so there should be a method to get accurate figures.  **How the limitations of the WCC metric be overcome**  Object-Oriented Software are based on classes, subclasses, and objects who are elements are methods and attributes. Therefore, special attention should be there on the methods and its attributes. Suitable weightage must be given for the methods as well as to the attributes and should categorize the methods as parameterized methods and non-parameterized methods. In addition to this, Cognitive functionalities are also very important in calculating the complexity of the code. It directly affects the readability of the code. Therefore, suitable cognitive weights should be assigned to each code line. Access modifiers also play a major role in the complexity of a method. There are four access modifiers namely public, private, protected, and default. Suitable weightage must be assigned to the access modifiers. WCC metric is too implementation technology-dependent therefore it should be able to give accurate outputs in any technology preferred. For this, a new metric should be introduced avoiding the given drawbacks of WCC and any object-oriented metrics introduced so far.  **How the limitations of the CFS metric be overcome**  This metric measures the cognitive functional size of a system. According to this metric, the sum of input and output and the total cognitive weight derives from the cognitive functional size. When considering the cognitive functional size metric, it also contains certain limitations. The cognitive functional size metric does not include information that is contained in operators and identifiers. “Implementation of CFS is easy and technology independent. It however excludes some essential details of cognitive complexity such as information that is contained in the identifiers and operators. It also does not consider some unique features of the object-oriented paradigm such as inheritance.” [3] The cognitive functional metric is designed to be used to calculate a method or function. If it is required to calculate for a whole software program, then it requires to calculate or each method and sum up all the values derived from the methods/ functions. So accordingly, the information that consists of inside operators and identifiers should be considered. Therefore, a new metric needs to be implemented to cover up all the drawbacks. |

* Brief explanation of the newly proposed complexity metric

|  |
| --- |
| Software complexity metrics are used to forecast important details regarding software systems' reliability and maintainability. Object-oriented metrics, paradigm-independent metrics, and procedure-oriented metrics, on the other hand, have several drawbacks now. In this context, the recently implemented complexity metrics attempt to avoid the pitfalls of those matrices and arrive at the best solution for calculating the code's complexity.  Cyclomatic Complexity -CC emphasizes more on the complexity of the control structures while ignoring the complexity of the data. Weighted Composite Complexity – WCC focuses more on the complexity of operands and operators rather than the internal architecture of the system. Cognitive Functional Size focuses more on the complexity of the basic control structures, but basic design aspects are equally important.  The newly proposed metrics utilize the factors such as the number of executable lines of code, inheritance levels including interfaces, types of access modifiers used in encapsulation, number of inputs and outputs in a code, number of nested levels, exception handling, method overloading, and overriding in polymorphism, number of parameters in a method, number of tokens, number of functions in a class, control structures and threads. The newly introduced metrics avoid the limitations of all currently available metrics. A programmer can reliably predict the complexity of any software or method using the new metrics equation and its derivation. |

* Factors considered by the new complexity metric. A group member should propose a **minimum** of **two factors**.

|  |
| --- |
| 1. Number of executable lines of code = 2. Inheritance Levels including interfaces = 3. Types of Access Modifiers used in Encapsulation = 4. Number of inputs and outputs in each class = 5. Nesting levels of Control Structures = 6. Exception Handling = 7. Overloading and Overriding in polymorphism = 8. Number of parameters in a method =   9. Token count = ***nT***   1. Number of functions in a class = 2. Control structures = 3. Number of Threads = |

* How the new metric captures the complexity introduced by each of its factors. If there are more than eight factors,
* add sub-headings accordingly.

|  |
| --- |
| 1. **How the new metric captures the complexity introduced due to *Number of executable lines of code*:**   Lines of Code (LOC) is an important property in measuring the size of a computer program. LOC is a way of measuring the size of the code by counting the number of instructions given in the code. While counting commented lines, header lines and white spaces are ignored. Measuring the size or the complexity of a program is not an easy task. Lines of executable codes can be used for this purpose. [4]  In this proposed metric we have developed an equation to calculate the number of executable lines of code.  The equation is as follows:  **= (Number of executable lines /Total Number of lines) \* 10**  **= Total number of executable lines of code**  To calculate the total number of executable lines of code, first, we must find the number of executable lines as given in the following code.  **Number of executable lines = Total number of lines - (Commented lines + White Space + Curly Braces)**  When we calculate the number of executable lines of code and divide it by the total number of lines, we will get a decimal value. The value obtained will be multiplied by 10 to convert it to a whole number. We could have straightaway counted the number of LOC without using an equation but then the complexity value will become too large if we have many lines of code. To avoid that above two equations are developed for the new metrics.   1. **How the new metric captures the complexity introduced due to *Inheritance Levels including interfaces*:**   The mechanism by which one class inherits the properties of another is known as inheritance. The use of such inheritance allows for the management of knowledge in a hierarchical order. A subclass is a class that adopts the characteristics of another, while a superclass is a type of whose characteristics are inherited. Programmers can render software more versatile by using inheritance inside it. By inheriting, child classes may use the parent class's functions and parameters, and certain properties of the superclass code, such as overriding superclass functions within the child class, can be added, or removed by the child classes.  Inheritance is identified by the newly proposed metrics as given below:  If the class is a super class/Parent class values are taken as 0.If the class is a child class is taken as 1. It will gradually increase by 1 when the number of inheritance level increases. If a class implements an interface the value will become . It will gradually increase when the number of interfaces increases.   * Parent/ Superclass -> * Child class **->** * Class inherits an interface ->  1. **How the new metric captures the complexity introduced due to the *Types of Access Modifiers used in Encapsulation*:**   In Java Encapsulation is done using Access Modifiers. As the name recommends access modifiers in Java assists with confining the extent of class, constructor, variable, method, or data member. There are four sorts of access modifiers accessible in java namely Private, Public, Protected, and Default.  Table  Description automatically generated  Encapsulation increases the readability while decreasing the code complexity.   * Public- Any Class can access this. * Protected – Can access from the Same class, Same package, Subclass of the same package, or a different package. * Private – Can access only from the same class. * Default – Default is when no modifier is specified. Can access inside the same package.   When a method or a data field is declared as private, it is hidden from the outside. This hiding enhances modularity while decreasing complexity.  When a method or a data field expose to the outside the complexity gradually increases. If the method/data field is public, a change done to it can affect every other class or method which uses that. This increases the complexity of the code.  In the proposed metric weight of 1 is assigned to private access modifiers while 2 is assigned on default. 3 will be assigned to protected and 4 will be assigned to public access modifiers as it is having the highest complexity.   * Private - 1 * Default – 2 * Protected – 3 * Public – 4  1. **How the new metric captures the complexity introduced due to *Number of inputs and outputs in each class*:**   The number of Inputs and Outputs in each class plays a major role in the complexity of the code. It is made up of input, output, and stream components. The data we provide to the system is the input. The data that we obtain from the program in the form of a result is referred to as the output. The flow of data, or the sequence of data, is represented by a stream. The input stream is used to provide input, and the output stream is used to provide output. Apart from CFS metrics, all the other metrics have not considered the number of inputs and outputs as a factor. The newly proposed metrics have considered this factor.  Number of Inputs:  Number of Outputs:  **=**  The number of inputs and outputs of the whole program has been considered here. Keyboards or File inputs are considered as input statements here while print statements and console outputs are taken as the output statements. For each Input and Output Statements, a value of 1 will be awarded.   1. **How the new metric captures the complexity introduced due to N*esting levels of Control Structures*:**   A control statement that is embedded inside another control statement is known as a nested control statement. You can take this to a lot of different levels, putting together complex coatings of different control statements. There are six control structures as block, while loop, do -while, for loop, if statements and switch statements. The compiler treats composite control statements as a single statement for the purpose of deciding what is influenced by the next higher degree of control.   * Sequential Statements = **= 0** * Statements inside the first level of control structures **= = 1** * Statements inside the second level of control structures **= = 2** * Statements inside the nth level of control structures **= = n**   When the nesting levels increases the value will increase accordingly.   1. **How the new metric captures the complexity introduced due to *Exception Handling*:**   Like control structures and functions in a program, exception handling is also a necessary component that should be used for a successful program to run properly. However, even when dealing with exception handling, the complexity of the program will be affected. Accordingly, when it comes to the complexity of exceptions they occur when they are an error in the program only. Exception handling codes are implemented using "throws" as throwing the exception and "try-catch" as catching the thrown exception and handling it. Throwing the exception does not affect the complexity of the program in a big way. So accordingly, the weight for throwing exception can be assigned as 1. Then the main part of handling exceptions is catching and handling the exception. For catch block of code, the weight could be assigned as 2 since it is the section when identifies what type of exception and what it needs to do next. So, after catching the exception the developer has the freedom to handle the error to which best for it.   1. **How the new metric captures the complexity introduced due to *Overloading and overriding in polymorphism*:**   Polymorphism is the main mechanism in object-oriented programming. It is enabled the ability to present the same class, method with several different data types. And it can uniquely respond to a different class, methods with the same content.  The main meaning of **overloading** is executing one or more methods with the same method name. But they should use different parameters, parameter type, and return type in the function. This overloading helps to code the reusability of the program. So, it is helping to reduce code complexity.  Also**overriding** a part of a polymorphism. Overriding is a feature in programming that allows subclass or child class to provide permission to implementation of a method or a function which already provided by its superclass or parent class. And overriding provide code reusability. So, it also helps to reduce code complexity.  In the proposed metric,  If the program or source code contains any overriding or overloading methods, that means the polymorphism is used in the program. Therefore,  **= Num of time overriding/overloading used in program**  **= Num of overriding/overloading \* - 1**  It is providing a negative number, Main reason for that polymorphism helps to reduce code complexity.   1. **How the new metric captures the complexity introduced due to *No of parameters in a method*:**   Any function/method can be categorized as parameterize and non-parameterize functions or methods. Therefore,  If a function or a method has may number of parameters, it will bring more complexity to a program or a source code. So, in the proposed metric,  Weight of a No of parameters in method calculates as follows,  **= No of parameters in a method**  **= No of parameters in a method \* 1**  If function is non-parameterize weight is taken as 0.   1. **How the new metric captures the complexity introduced due to** ***Token count***:   **The token count** is a count of tokens in a source code or a program. A token is a simple entity that makes up a source code or a program. The token is categorized as **operators** or **operands**.  **Operators** are known as arithmetic symbols such as +, -, /, \* and command names such as “if”, “for”, “printf”, special symbols such as: , =, braces and parentheses, and even function names.  **Operands**are known as all symbols used to represent data like variables, constants, and even labels.  whenever Token counts are high in the program automatically complexity will be high.  Therefore, the token count will calculate as follows,  ***nT* = Number of Unique operators + Number of Unique Operands**  token count always will be positive value.   1. **How the new metric captures the complexity introduced due to the *Number of functions in class*:**   When it comes to number of functions, I would like to provide 1 as a weight to a function with no input parameters and void output, for no of parameterized functions, for each parameter 1 weight is assigned. For example, if a function has 2 parameters and void as return type then that function has 3 as weight. When a function has return type as primitive types like (integer, double, string, Boolean) that is also counted as 1 but if the return type is references type like returning an object, then the weight would be 2, if it is a void return type it is considered as 0 weight. In a class if the number of functions is 2 where one function has no parameter and no return type so the weight will be considered as 1 and another function with 1 parameter and double return type Boolean, then the complexity would be,  Function 1 = weight with no parameter and return type = 1  Function 2,  Weight of parameter = 1  Weight of output = 1  total = 1 +1 = 2  Total complexity of both functions = 2 + 1 = 3  Therefore, the complexity of the program or class depends on the number of functions used and the type of functions used.   1. **How the new metric captures the complexity introduced due to the *Control Structures*:**   Control structures play a vital role when it comes to developing software applications. The complexity of the control structure is very crucial when using them. Accordingly, 7 basic control structures will be considered here, they are sequence, branch, loop, function calls, recursion, interrupt /break and continue. Based on the type of control structures, their weights would be assigned.  - Sequence category  The function of sequence control structure is to execute code sequentially/ in order; therefore the weight will be assigned as 1.  - Branch category  For branch category, (If-else) the weight will be assigned as 2 only if there is a single condition. if there are an If-else statement that has "&&" or "||" operators, then for each set of operators 1 weight will be added. for switch-case statements, the weight will be assigned based on the number of cases. If there are 'n' number of cases, then the weight will be added "n" number of times.  - Iteration Category  According to the Iteration category, there are three control structures as while, for, and do-while. The weight will be assigned as 3 for them, for variable assignment = 1, increment = 1, condition checking = 1.  - Function call control  In function call control structure, the weight will be assigned as 2, since when it meets a function call it must find the function and return to the function either with a value or without a value. If the function has a parameter, the weight will consider as 3.  - Interrupt/break control structure.  According to this control structure, it is mostly used in looping/ iteration statements to stop the execution and jump out of the loop. Based on that the weight is assigned as 2.  - continue control structure  Same as interrupt/ break control structure, this control structure is also used mostly in iterative statements, but the function of this is when the condition meets true then that executable statement will be skipped and continues from the next flow of execution.  However, using the basic control is necessary, but when using the control structures, it is also necessary to consider the complexity of the program.   1. **How the new metric captures the complexity introduced due to the *Number of Threads*:**   Inside a system, a thread is a single sequential flow of command. The essential thrill of threads is not centered on a single sequential thread. Indeed, it is about using numerous threads to do multiple activities in a single file at the same time. A thread is a lightweight process of a program, thus When threads are running, they use shared memory location. The main thing of thread is they are doing different tasks concurrently. As an example, if we want to build a program to connect with other computers and get some data. When such situations we can implement two threads. One for maintaining the connection and the other for communication. Either we define or not main method runs on the main thread in java. Therefore,  **= Number of threads used in program**  A value of 1 will be awarded for each thread. |

* Complexity calculation formula/equation of the new metric. Mention the meanings of all the characters/symbols used in the formula/equation.

|  |
| --- |
| **Complexity = T \* + +**  **= + + + +**  **= + + + +**   1. Line of executable codes = 2. Inheritance Levels including interfaces = 3. Types of Access Modifiers used in Encapsulation = 4. No of inputs and outputs in each class = 5. Nesting levels of Control Structures = 6. Exception Handling = 7. Overloading and Overriding in polymorphism = 8. No of parameters in a method = 9. Token count = ***nT*** 10. No of functions in a class = 11. Control structures = 12. Number of Threads =   **= Summation of inheritance levels, number of access modifiers, nesting levels of Control Structures, number of parameters and number of control structures.**  **=**  **= Addition of number of inputs and outputs, exceptions, number of functions and number of threads and subtracting the overriding and overloading methods from it.**  **= (Total Number of executable lines/Total Number of lines) \* 10**  **Total Number of executable lines = Total number of lines - (Commented lines + White Space + Curly Braces)** |

* Rationale behind the complexity calculation formula/equation of the new metric

|  |
| --- |
| **Complexity = T \* + +**  To calculate the complexity of a metric, we must multiply the total number of token counts-nT by and add to it.  **= + + + +**  **= Summation of inheritance levels, number of access modifiers, nesting levels of Control Structures, number of parameters and number of control structures.**  Inheritance levels, number of access modifiers, nested levels, number of parameters, and number of control structures are considered line by line therefore those factors have been multiplied by the token count to get the product.  **=**  **= Addition of number of inputs and outputs, exceptions, number of functions and number of threads and subtracting the overriding and overloading methods from it.**  **= (Total Number of executable lines/Total Number of lines) \* 10**  **Total Number of executable lines = Total number of lines - (Commented lines + White Space + Curly Braces)**  values are not considered line by line therefore no need to multiply them by the token count. Those two values are obtained from the outside and added to the final equation as given above.  which is the total number of executed lines will give a decimal value and to remove the decimal place it is multiplied by 10. is increasing the complexity and that is the reason why we must add that value to the final equation.  Factors considered under is also increasing the code complexity therefore they should be added to the final equation. |

* Calculation of complexity of the first java program using the newly proposed metric

|  |
| --- |
| Source code of the first java program  public class Vehicle {  //Attributes  private String Tyres;  private String[] Lights;  private String BodyColor;  private int NoOfTyres;    //Attributes for pricing  private double TotalPriceOfTyres;  private double TotalPriceOfLights;  private double TotalPriceOfBodyColor;    public Vehicle(String tyres, int NoOfTyres,String[] lights, String bodyColor) {  Tyres = tyres;  Lights = lights;  this.NoOfTyres = NoOfTyres;  BodyColor = bodyColor;    this.TotalPriceOfTyres = 0;  this.TotalPriceOfLights = 0;  this.TotalPriceOfBodyColor = 0;  }  public String getTyres() {  return Tyres;  }  public void setTyres(String tyres) {  Tyres = tyres;  }  public String[] getLights() {  return Lights;  }  public void setLights(String[] lights) {  Lights = lights;  }  public String getBodyColor() {  return BodyColor;  }  public void setBodyColor(String bodyColor) {  BodyColor = bodyColor;  }    private void SetValuesForPricing() {  double Amount = 0;    //set amount for tyres  switch (this.Tyres.toUpperCase()) {  case "MRF":  if(NoOfTyres > 0) {  TotalPriceOfTyres = 20000 \* NoOfTyres;  }  break;  case "DSI":  if(NoOfTyres > 0) {  TotalPriceOfTyres = 15000 \* NoOfTyres;  }  break;  default:  if(NoOfTyres > 0) {  TotalPriceOfTyres = 10000 \* NoOfTyres;  }  break;  }    //for check lights and set amount  try {  if(Lights.length > 1) {  for(String li : Lights) {  switch (li.toUpperCase()) {  case "BREAKLIGHT":  Amount += 5500;  break;  case "HEADLIGHT":  Amount += 15000;  break;  default:  break;  }  }  TotalPriceOfLights = Amount;  }  } catch (Exception e) {  e.printStackTrace();  }    //check request body color and get amount  if(!BodyColor.equalsIgnoreCase("")) {  switch (BodyColor.toUpperCase()) {  case "WHITE":  TotalPriceOfBodyColor = 100000;  break;  case "BLACK":  TotalPriceOfBodyColor = 200000;  break;  case "RED":  TotalPriceOfBodyColor = 300000;  break;  default:  break;  }  }    }    //calculate and print receipt without discount  public void CalculateAndPrintReceipt() {  SetValuesForPricing();  System.out.println();  System.out.println(">>>>>>>>>>> Receipt <<<<<<<<<<<");  System.out.println();  double TotalAmout = this.TotalPriceOfBodyColor + this.TotalPriceOfLights + this.TotalPriceOfTyres;  System.out.println("Total Amout :" + TotalAmout);  }    //calculate and print receipt with discount  public void CalculateAndPrintReceipt(double discount) {  SetValuesForPricing();  System.out.println();  System.out.println(">>>>>>>>>>> Receipt <<<<<<<<<<<");  System.out.println();  double TotalAmout = this.TotalPriceOfBodyColor + this.TotalPriceOfLights + this.TotalPriceOfTyres;  System.out.println("Total Amout :" + (TotalAmout - discount));  }  }  public class LightVehicle extends Vehicle implements Modify,Cleaning{  private double Discount;  private boolean EnableModify = false;  private boolean EnableCleaning = false;  public LightVehicle(String tyres, int NoOfTyres, String[] lights, String bodyColor, double discount,  boolean enableModify, boolean enableCleaning) {  super(tyres, NoOfTyres, lights, bodyColor);  Discount = discount;  EnableModify = enableModify;  EnableCleaning = enableCleaning;  }    //this method controls all methods according to user's request  public void StartService() {  System.out.println("Welcome To ABC Service");  if(EnableModify) {  StartModifingVehicle();  }    if(EnableCleaning) {  StartCleaningVehicle();  }    CalculateAndPrintReciept();  }      //for clean a vehicle  @Override  public void StartCleaningVehicle() {  System.out.println();  for(int i = 0; i < 10; i++) {  try {  System.out.println(">>>>>>>>>Cleaning Your Vehicle <<<<<<<<<<<<<");  Thread.sleep(1000);  } catch (Exception e) {  e.printStackTrace();  }  }  System.out.println();  System.out.println(">>>>>>>>> Completed Cleaning Task <<<<<<<<<<<<<");  }  //for start modify a vehicle  @Override  public void StartModifingVehicle() {  System.out.println();  for(int i = 0; i < 10; i++) {  try {  System.out.println(">>>>>>>>>Modifing Your Vehicle <<<<<<<<<<<<<");  Thread.sleep(1000);  } catch (Exception e) {  e.printStackTrace();  }  }  System.out.println();  System.out.println(">>>>>>>>> Completed Modifying Task <<<<<<<<<<<<<");  System.out.println();    }  //print receipt  public void CalculateAndPrintReciept() {  if(this.Discount > 0) {  super.CalculateAndPrintReceipt(Discount);  }else {  super.CalculateAndPrintReceipt();  }  }  }  public interface Modify {  public void StartModifingVehicle();  }  public interface Cleaning {  public void StartCleaningVehicle();  }  import java.util.Scanner;  public class VehicleMain {  public static void main(String[] args) {  Scanner scn = new Scanner(System.in);    String[] li = {"BREAKLIGHT","HEADLIGHT"};    System.out.println(">>>>>>>>>Welcome to service Application<<<<<<<<<");    //get tyre type  System.out.print("Enter Tyre Types : ");  String Tyre = scn.next();    //get tyre count  System.out.print("Enter No Of Tyres You Want : ");  int TyreCount = scn.nextInt();    //get lights  String[] lights = new String[li.length];  System.out.println("Select Lights ");  for(int i = 0; i < li.length ; i++) {  System.out.print("Do you want add new " + li[i] + " . press y to select and press any key to move next : ");  String res = scn.next();  if(res.toUpperCase().equals("y")) {  lights[i] = li[i];  }  System.out.println();  }    //get body colour  System.out.print("Enter body color : ");  String BodyColor = scn.next();    System.out.println();  System.out.println();    LightVehicle v1 = new LightVehicle(Tyre, TyreCount, lights, BodyColor, 1500, true, true);  v1.StartService();  }  } |
| Calculation of complexity of the first java program  Calculating the Token Count -**nT**  Table 1   |  |  |  |  | | --- | --- | --- | --- | | **Line No** | **Program Statements** | **Tokens** | **nT(Token Count)** | | 1. | public class Vehicle { |  |  | | 2. | private String Tyres; | String, Tyres | 2 | | 3. | private String[] Lights; | String[], Lights | 2 | | 4. | private String BodyColor; | String, BodyColor | 2 | | 5. | private int NoOfTyres; | Int, NoOfTyres | 2 | | 6. | private double TotalPriceOfTyres; | double, TotalPriceOfTyres | 2 | | 7. | private double TotalPriceOfLights; | double, TotalPriceOfLights | 2 | | 8. | private double TotalPriceOfBodyColor; | double, TotalPriceOfBodyColor | 2 | | 9. | public Vehicle(String tyres, int NoOfTyres,String[] lights, String bodyColor) { | Vehicle(), String, tyres, int, NoOfTyres, String[], lights, String, bodyColor | 9 | | 10. | Tyres = tyres; | Tyres, = , tyres | 3 | | 11. | Lights = lights; | Lights, =, lights | 3 | | 12. | this.NoOfTyres = NoOfTyres; | this, . , NoOfTyres, =, NoOfTyres | 5 | | 13. | BodyColor = bodyColor; | BodyColor, =, bodyColor | 3 | | 14. | this.TotalPriceOfTyres = 0; | this, ., TotalPriceOfTyres, =, 0 | 5 | | 15. | this.TotalPriceOfLights = 0; | this, ., TotalPriceOfLights, =, 0 | 5 | | 16. | this.TotalPriceOfBodyColor = 0; | this, ., TotalPriceOfBodyColor, =, 0 | 5 | | 17. | } |  |  | | 18. | public String getTyres() { | String, getTyres() | 2 | | 19. | return Tyres; | return, Tyres | 2 | | 20. | } |  |  | | 21. | public void setTyres(String tyres) { | void, setTyres(), String, tyres | 4 | | 22. | Tyres = tyres; | Tyres, =, tyres | 3 | | 23. | } |  |  | | 24. | public String[] getLights() { | String[],getLights() | 2 | | 25. | return Lights; | return, Lights | 2 | | 26. | } |  |  | | 27. | public void setLights(String[] lights) { | void, setLights(),String[],lights | 4 | | 28. | Lights = lights; | Lights, =, lights | 3 | | 29. | } |  |  | | 30. | public String getBodyColor() { | String, getBodyColor() | 2 | | 31. | return BodyColor; | return, BodyColor | 2 | | 32. | } |  |  | | 33. | public void setBodyColor(String bodyColor) { | void, setBodyColor(), String, bodyColor | 4 | | 34. | BodyColor = bodyColor; | BodyColor, =, bodyColor | 3 | | 35. | } |  |  | | 36. | private void SetValuesForPricing() { | void, SetValuesForPricing() | 2 | | 37. | double Amount = 0; | double, Amount, =, 0 | 4 | | 38. | switch (this.Tyres.toUpperCase()) { | switch(), this, . , Tyres, toUpperCase() | 5 | | 39. | case "MRF": | case“”, MRF, : | 3 | | 40. | if(NoOfTyres > 0) { | if(), NoOfTyres, >, 0 | 4 | | 41. | TotalPriceOfTyres = 20000 \* NoOfTyres; | TotalPriceOfTyres , =, 20000, \* , NoOfTyres | 5 | | 42. | } |  |  | | 43. | break; | break | 1 | | 44. | case "DSI": | case“”, DSI, : | 3 | | 45. | if(NoOfTyres > 0) { | if(), NoOfTyres, >, 0 | 4 | | 46. | TotalPriceOfTyres = 15000 \* NoOfTyres; | TotalPriceOfTyres, =, 15000, \*, NoOfTyres | 5 | | 47. | } |  |  | | 48. | break; | break | 1 | | 49. | default: | default, : | 2 | | 50. | if(NoOfTyres > 0) { | if(), NoOfTyres, >, 0 | 4 | | 51. | TotalPriceOfTyres = 10000 \* NoOfTyres; | TotalPriceOfTyres, =, 10000, \*, NoOfTyres | 5 | | 52. | } |  |  | | 53. | break; | break | 1 | | 54. | } |  |  | | 62. | try { |  |  | | 63. | if(Lights.length > 1) { | if(), Lights, ., length, >, 1 | 6 | | 64. | for(String li : Lights) { | for(), String, li, :, Lights | 5 | | 65. | switch (li.toUpperCase()) { | switch(), li, ., toUpperCase() | 4 | | 66. | case "BREAKLIGHT": | case””, BREAKLIGHT, : | 3 | | 67. | Amount += 5500; | Amount, +, =, 5500 | 4 | | 68. | break; | break | 1 | | 69. | case "HEADLIGHT": | case””, HEADLIGHT, : | 3 | | 70. | Amount += 15000; | Amount , +, =, 15000 | 4 | | 71. | break; | break | 1 | | 72. | default: | default, : | 2 | | 73. | break; | break | 1 | | 74. | } |  |  | | 75. | } |  |  | | 76. | } catch (Exception e) { | catch(), Exception, e | 3 | | 77. | e.printStackTrace(); | e, . , printStackTrace() | 3 | | 78. | } |  |  | | 79. | if(!BodyColor.equalsIgnoreCase("")) { | if(), !, BodyColor, . , equalsIgnoreCase(), “” | 6 | | 80. | switch (BodyColor.toUpperCase()) { | switch, BodyColor, . , toUpperCase() | 4 | | 81. | case "WHITE": | case””, WHITE, : | 3 | | 82. | TotalPriceOfBodyColor = 100000; | TotalPriceOfBodyColor, =, 100000 | 3 | | 83. | break; | break | 1 | | 84. | case "BLACK": | case””, BLACK, : | 3 | | 85. | TotalPriceOfBodyColor = 200000; | TotalPriceOfBodyColor, =, 200000 | 3 | | 86. | break; | break | 1 | | 87. | case "RED": | case””, RED, : | 3 | | 88. | TotalPriceOfBodyColor = 300000; | TotalPriceOfBodyColor, = , 300000 | 4 | | 89. | break; | break | 1 | | 90. | default: | default, : | 2 | | 91. | break; | break | 1 | | 92. | } |  |  | | 93. | } |  |  | | 94. | } |  |  | | 95. | public void CalculateAndPrintReceipt() { | void, CalculateAndPrintReceipt() | 2 | | 96. | SetValuesForPricing(); | SetValuesForPricing() | 1 | | 97. | System.out.println(); | System, ., out, . , println() | 5 | | 98. | System.out.println(">>>>>>>>>>> Receipt <<<<<<<<<<<"); | System, ., out, . , println() , ">>>>>>>>>>> Receipt <<<<<<<<<<<" | 6 | | 99. | System.out.println(); | System, ., out, . , println() | 5 | | 100. | double TotalAmout = this.TotalPriceOfBodyColor + this.TotalPriceOfLights + this.TotalPriceOfTyres; | double, TotalAmout, =, this, . , TotalPriceOfBodyColor, + , this, . , TotalPriceOfLights , + , this, . , TotalPriceOfTyres | 15 | | 101. | System.out.println("Total Amout :" + TotalAmout); | System, ., out, . , println(), "Total Amout :", + , TotalAmout | 8 | | 102. | } |  |  | | 103. | public void CalculateAndPrintReceipt(double discount) { | void, CalculateAndPrintReceipt, double, discount | 4 | | 104. | SetValuesForPricing(); | SetValuesForPricing() | 1 | | 105. | System.out.println(); | System, ., out, . , println() | 5 | | 106. | System.out.println(">>>>>>>>>>> Receipt <<<<<<<<<<<"); | System, ., out, . , println(), ">>>>>>>>>>> Receipt <<<<<<<<<<<" | 6 | | 107. | System.out.println(); | System, ., out, . , println() | 5 | | 108. | double TotalAmout = this.TotalPriceOfBodyColor + this.TotalPriceOfLights + this.TotalPriceOfTyres; | double, TotalAmout, =, this, ., TotalPriceOfBodyColor, + , this, . , TotalPriceOfLights, + , this , . , TotalPriceOfTyres | 15 | | 109. | System.out.println("Total Amout :" + (TotalAmout - discount)); | System, ., out, . , println(),"Total Amout :" , + , () , TotalAmout, - , discount | 11 | | 110. | } |  |  | | 111. | } |  |  | | 112. | public class LightVehicle extends Vehicle implements Modify,Cleaning{ |  |  | | 113. | private double Discount; | double, Discount | 2 | | 114. | private boolean EnableModify = false; | boolean, EnableModify, =, false | 4 | | 115. | private boolean EnableCleaning = false; | boolean, EnableCleaning, =, false | 4 | | 116. | public LightVehicle(String tyres, int NoOfTyres, String[] lights, String bodyColor, double discount, boolean enableModify, boolean enableCleaning) { | LightVehicle(), String, tyres, int, NoOfTyers, String[], lights, String, bodycolor, double, discount, boolean, enableModify, boolean, enableCleaning | 15 | | 117. | super(tyres, NoOfTyres, lights, bodyColor); | Super(), tyres, NoOfTyres, lights, bodyColor | 5 | | 118. | Discount = discount; | Discount, =, discount | 3 | | 119. | EnableModify = enableModify; | EnableModify, =, enableModify | 3 | | 120. | EnableCleaning = enableCleaning; | EnableCleaning, =, enableCleaning | 3 | | 121. | } |  |  | | 122. | public void StartService() { | Void, StartService() | 2 | | 123. | System.out.println("Welcome To ABC Service"); | System, ., out, . , println() | 5 | | 124. | if(EnableModify) { | If(), EnableModify | 2 | | 125. | StartModifingVehicle(); | StartModifingVehicle() | 1 | | 126. | } |  |  | | 127. | if(EnableCleaning) { | If(),EnableCleaning | 2 | | 128. | StartCleaningVehicle(); | StartCleaningVehicle() | 1 | | 129. | } |  |  | | 130. | CalculateAndPrintReciept(); | CalculateAndPrintReciept() | 1 | | 131. | } |  |  | | 132. | public void StartCleaningVehicle() { | void, StartCleaningVehicle | 2 | | 133. | System.out.println(); | System, ., out, . , println() | 5 | | 134. | for(int i = 0; i < 10; i++) { | for(), int, I, =, 0, <, 10, + | 8 | | 135. | try { |  |  | | 136. | System.out.println(">>>>>>>>>Cleaning Your Vehicle <<<<<<<<<<<<<"); | System, ., out, . , println(),">>>>>>>>>Cleaning Your Vehicle <<<<<<<<<<<<<" | 6 | | 137. | Thread.sleep(1000); | Thread, . , sleep(), 1000 | 4 | | 138. | } catch (Exception e) { | catch(), Exception, e | 3 | | 139. | e.printStackTrace(); | e, printStackTrace() | 2 | | 140. | } |  |  | | 141. | } |  |  | | 142. | System.out.println(); | System, ., out, . , println() | 5 | | 143. | System.out.println(">>>>>>>>> Completed Cleaning Task <<<<<<<<<<<<<"); | System, ., out, . , println() , ">>>>>>>>> Completed Cleaning Task <<<<<<<<<<<<<" | 6 | | 144. | } |  |  | | 145. | public void StartModifingVehicle() { | void, StartModifingVehicle() | 2 | | 146. | System.out.println(); | System, ., out, . , println() | 5 | | 147. | for(int i = 0; i < 10; i++) { | for(), int, I, =, 0, <, 10, + | 8 | | 148. | try { |  |  | | 149. | System.out.println(">>>>>>>>>Modifing Your Vehicle <<<<<<<<<<<<<"); | System, ., out, . , println(),">>>>>>>>>Modifing Your Vehicle <<<<<<<<<<<<<" | 6 | | 150. | Thread.sleep(1000); | Thread, . , sleep(), 1000 | 4 | | 151. | } catch (Exception e) { | catch(), Exception, e | 3 | | 152. | e.printStackTrace(); | e, printStackTrace() | 2 | | 153. | } |  |  | | 154. | } |  |  | | 155. | System.out.println(); | System, ., out, . , println() | 5 | | 156. | System.out.println(">>>>>>>>> Completed Modifying Task <<<<<<<<<<<<<"); | System, ., out, . , println(), ">>>>>>>>> Completed Modifying Task <<<<<<<<<<<<<" | 6 | | 157. | System.out.println(); | System, ., out, . , println() | 5 | | 158 | } |  |  | | 159. | public void CalculateAndPrintReciept() { | void, CalculateAndPrintReciept() | 2 | | 160. | if(this.Discount > 0) { | If(), this, ., Discount, >, 0 | 6 | | 161. | super.CalculateAndPrintReceipt(Discount); | Super, ., CalculateAndPrintReceipt(), Discount | 5 | | 162. | }else { |  |  | | 163. | super.CalculateAndPrintReceipt(); | Super, ., CalculateAndPrintReceipt() | 4 | | 164. | } |  |  | | 165. | } |  |  | | 166. | } |  |  | | 167. | public interface Modify { |  |  | | 168. | public void StartModifingVehicle(); | void, StartModifingVehicle() | 2 | | 169. | } |  |  | | 170. | public interface Cleaning { |  |  | | 171. | public void StartCleaningVehicle(); | void, StartCleaningVehicle() | 2 | | 172. | } |  |  | | 173. | import java.util.Scanner; |  |  | | 174. | public class VehicleMain { |  |  | | 175. | public static void main(String[] args) { | static, void, main(), String[], args | 5 | | 176. | Scanner scn = new Scanner(System.in); | Scanner, scn, = , new, Scanner(), System, ., in | 8 | | 177. | String[] li = {"BREAKLIGHT","HEADLIGHT"}; | String[], li, = , {}, "BREAKLIGHT"," HEADLIGHT" | 6 | | 178. | System.out.println(">>>>>>>>>Welcome to service Application<<<<<<<<<"); | System, ., out, . , println(),">>>>>>>>>Welcome to service Application<<<<<<<<<" | 6 | | 179. | System.out.print("Enter Tyre Types : "); | System, ., out, . , println(), “Enter Tyre Types : " | 6 | | 180. | String Tyre = scn.next(); | String , Tyre, = , scn, ., next() | 6 | | 181. | System.out.print("Enter No Of Tyres You Want : "); | System, ., out, . , println() , "Enter No Of Tyres You Want : " | 6 | | 182. | int TyreCount = scn.nextInt(); | int, TyreCount, = , scn, nextInt() | 5 | | 183. | String[] lights = new String[li.length]; | String[], lights, =, new, String[], li, ., lenght | 8 | | 184. | System.out.println("Select Lights "); | System, ., out, . , println(), "Select Lights " | 6 | | 185. | for(int i = 0; i < li.length ; i++) { | for(), int, =, 0, I, <, 10, + | 8 | | 186. | System.out.print("Do you want add new " + li[i] + " . press y to select and press any key to move next : "); | System, ., out, . , println(), "Do you want add new " , + , li[i], +, " . press y to select and press any key to move next :" | 10 | | 187. | String res = scn.next(); | String, res, =, scn, next() | 5 | | 188. | if(res.toUpperCase().equals("y")) { | If(), res, . , toUpperCase(), ., equals(), “y” | 7 | | 189. | lights[i] = li[i]; | lights[i], = , li[i] | 3 | | 190. | } |  |  | | 191. | System.out.println(); | System, ., out, . , println() | 5 | | 192. | } |  |  | | 193. | System.out.print("Enter body color : "); | System, ., out, . , println(), “Enter body color :” | 6 | | 194. | String BodyColor = scn.next(); | String, BodyColor, =, scn, ., next() | 6 | | 195. | System.out.println(); | System, ., out, . , println() | 5 | | 196. | System.out.println(); | System, ., out, . , println() | 5 | | 197. | LightVehicle v1 = new LightVehicle(Tyre, TyreCount, lights, BodyColor, 1500, true, true); | LightVehicle, v1, =, new, LightVehicle(), Tyre, TyreCount, lights, BodyColor, 1500, true, true | 12 | | 198. | v1.StartService(); | v1, . , StartService() | 3 | | 199. | } |  |  | | 200. | } |  |  |   Calculating nT \*  Table 2   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Line No** | **Program Statements** | **nT(Token Count)** |  |  |  |  |  |  | **nT\*** | | 1. | public class Vehicle { |  |  |  |  |  |  |  |  | | 2. | private String Tyres; | 2 | 0 | 1 | 0 | 0 | 0 | 1 | 2 | | 3. | private String[] Lights; | 2 | 0 | 1 | 0 | 0 | 0 | 1 | 2 | | 4. | private String BodyColor; | 2 | 0 | 1 | 0 | 0 | 0 | 1 | 2 | | 5. | private int NoOfTyres; | 2 | 0 | 1 | 0 | 0 | 0 | 1 | 2 | | 6. | private double TotalPriceOfTyres; | 2 | 0 | 1 | 0 | 0 | 0 | 1 | 2 | | 7. | private double TotalPriceOfLights; | 2 | 0 | 1 | 0 | 0 | 0 | 1 | 2 | | 8. | private double TotalPriceOfBodyColor; | 2 | 0 | 1 | 0 | 0 | 0 | 1 | 2 | | 9. | public Vehicle(String tyres, int NoOfTyres,String[] lights, String bodyColor) { | 9 | 0 | 4 | 0 | 4 | 0 | 8 | 72 | | 10. | Tyres = tyres; | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 11. | Lights = lights; | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 12. | this.NoOfTyres = NoOfTyres; | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 13. | BodyColor = bodyColor; | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 14. | this.TotalPriceOfTyres = 0; | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 15. | this.TotalPriceOfLights = 0; | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 16. | this.TotalPriceOfBodyColor = 0; | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 17. | } |  |  |  |  |  |  |  |  | | 18. | public String getTyres() { | 2 | 0 | 4 | 0 | 0 | 0 | 0 | 0 | | 19. | return Tyres; | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 20. | } |  |  |  |  |  |  |  |  | | 21. | public void setTyres(String tyres) { | 4 | 0 | 4 | 0 | 1 | 0 | 5 | 20 | | 22. | Tyres = tyres; | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 23. | } |  |  |  |  |  |  |  |  | | 24. | public String[] getLights() { | 2 | 0 | 4 | 0 | 0 | 0 | 4 | 8 | | 25. | return Lights; | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 26. | } |  |  |  |  |  |  |  |  | | 27. | public void setLights(String[] lights) { | 4 | 0 | 4 | 0 | 1 | 0 | 5 | 20 | | 28. | Lights = lights; | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 29. | } |  |  |  |  |  |  |  |  | | 30. | public String getBodyColor() { | 2 | 0 | 4 | 0 | 0 | 0 | 4 | 8 | | 31. | return BodyColor; | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 32. | } |  |  |  |  |  |  |  |  | | 33. | public void setBodyColor(String bodyColor) { | 4 | 0 | 4 | 0 | 1 | 0 | 5 | 20 | | 34. | BodyColor = bodyColor; | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 35. | } |  |  |  |  |  |  |  |  | | 36. | private void SetValuesForPricing() { | 2 | 0 | 1 | 0 | 0 | 0 | 1 | 2 | | 37. | double Amount = 0; | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 38. | switch (this.Tyres.toUpperCase()) { | 5 | 0 | 0 | 1 | 0 | 2 | 3 | 15 | | 39. | case "MRF": | 3 | 0 | 0 | 1 | 0 | 0 | 1 | 3 | | 40. | if(NoOfTyres > 0) { | 4 | 0 | 0 | 2 | 0 | 2 | 4 | 16 | | 41. | TotalPriceOfTyres = 20000 \* NoOfTyres; | 5 | 0 | 0 | 2 | 0 | 0 | 2 | 10 | | 42. | } |  |  |  |  |  |  |  |  | | 43. | break; | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | | 44. | case "DSI": | 3 | 0 | 0 | 0 | 0 | 0 | 1 | 3 | | 45. | if(NoOfTyres > 0) { | 4 | 0 | 0 | 0 | 0 | 2 | 4 | 16 | | 46. | TotalPriceOfTyres = 15000 \* NoOfTyres; | 5 | 0 | 0 | 0 | 0 | 0 | 2 | 10 | | 47. | } |  |  |  |  |  |  |  |  | | 48. | break; | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | | 49. | default: | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | | 50. | if(NoOfTyres > 0) { | 4 | 0 | 0 | 0 | 0 | 2 | 4 | 16 | | 51. | TotalPriceOfTyres = 10000 \* NoOfTyres; | 5 | 0 | 0 | 0 | 0 | 0 | 2 | 10 | | 52. | } |  |  |  |  |  |  |  |  | | 53. | break; | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 54. | } |  |  |  |  |  |  |  |  | | 55. | try { |  |  |  |  |  |  |  |  | | 56. | if(Lights.length > 1) { | 6 | 0 | 0 | 0 | 0 | 2 | 3 | 18 | | 57. | for(String li : Lights) { | 5 | 0 | 0 | 0 | 0 | 3 | 5 | 25 | | 58. | switch (li.toUpperCase()) { | 4 | 0 | 0 | 0 | 0 | 2 | 5 | 20 | | 59. | case "BREAKLIGHT": | 3 | 0 | 0 | 0 | 0 | 0 | 3 | 9 | | 60. | Amount += 5500; | 4 | 0 | 0 | 0 | 0 | 0 | 3 | 12 | | 61. | break; | 1 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | | 62 | case "HEADLIGHT": | 3 | 0 | 0 | 0 | 0 | 0 | 3 | 9 | | 63. | Amount += 15000; | 4 | 0 | 0 | 0 | 0 | 0 | 3 | 12 | | 64. | break; | 1 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | | 65. | default: | 2 | 0 | 0 | 0 | 0 | 0 | 3 | 6 | | 66. | break; | 1 | 0 | 0 | 0 | 0 | 0 | 3 | 3 | | 67. | } |  |  |  |  |  |  |  |  | | 68. | } |  |  |  |  |  |  |  |  | | 69. | } catch (Exception e) { | 1 | 0 | 0 | 1 | 1 | 0 | 2 | 2 | | 70. | e.printStackTrace(); | 3 | 0 | 0 | 0 | 0 | 0 | 1 | 3 | | 71. | } |  |  |  |  |  |  |  |  | | 72. | if(!BodyColor.equalsIgnoreCase("")) { | 6 | 0 | 0 | 0 | 0 | 2 | 3 | 18 | | 73. | switch (BodyColor.toUpperCase()) { | 4 | 0 | 0 | 0 | 0 | 3 | 5 | 20 | | 74. | case "WHITE": | 3 | 0 | 0 | 0 | 0 | 0 | 2 | 6 | | 75. | TotalPriceOfBodyColor = 100000; | 3 | 0 | 0 | 0 | 0 | 0 | 2 | 6 | | 76. | break; | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | | 77. | case "BLACK": | 3 | 0 | 0 | 0 | 0 | 0 | 2 | 6 | | 78. | TotalPriceOfBodyColor = 200000; | 3 | 0 | 0 | 0 | 0 | 0 | 2 | 6 | | 79. | break; | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | | 80. | case "RED": | 3 | 0 | 0 | 0 | 0 | 0 | 2 | 6 | | 81. | TotalPriceOfBodyColor = 300000; | 4 | 0 | 0 | 0 | 0 | 0 | 2 | 8 | | 82. | break; | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | | 83. | default: | 2 | 0 | 0 | 0 | 0 | 0 | 2 | 4 | | 84. | break; | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 2 | | 85. | } |  |  |  |  |  |  |  |  | | 86. | } |  |  |  |  |  |  |  |  | | 87. | } |  |  |  |  |  |  |  |  | | 88. | public void CalculateAndPrintReceipt() { | 2 | 0 | 4 | 0 | 0 | 0 | 4 | 8 | | 89. | SetValuesForPricing(); | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 90. | System.out.println(); | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 91. | System.out.println(">>>>>>>>>>> Receipt <<<<<<<<<<<"); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 92. | System.out.println(); | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 93. | double TotalAmout = this.TotalPriceOfBodyColor + this.TotalPriceOfLights + this.TotalPriceOfTyres; | 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 94. | System.out.println("Total Amout :" + TotalAmout); | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 95. | } |  |  |  |  |  |  |  |  | | 96. | public void CalculateAndPrintReceipt(double discount) { | 4 | 0 | 4 | 1 | 1 | 0 | 6 | 24 | | 97. | SetValuesForPricing(); | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 98. | System.out.println(); | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 99. | System.out.println(">>>>>>>>>>> Receipt <<<<<<<<<<<"); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 100. | System.out.println(); | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 101. | double TotalAmout = this.TotalPriceOfBodyColor + this.TotalPriceOfLights + this.TotalPriceOfTyres; | 15 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 102. | System.out.println("Total Amout :" + (TotalAmout - discount)); | 11 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 103. | } |  |  |  |  |  |  |  |  | | 104. | } |  |  |  |  |  |  |  |  | | 105. | public class LightVehicle extends Vehicle implements Modify,Cleaning{ |  | 3 | 4 | 0 | 0 | 0 | 7 | 0 | | 106. | private double Discount; | 2 | 3 | 1 | 0 | 0 | 0 | 4 | 8 | | 107. | private boolean EnableModify = false; | 4 | 3 | 1 | 0 | 0 | 0 | 4 | 16 | | 108. | private boolean EnableCleaning = false; | 4 | 3 | 1 | 0 | 0 | 0 | 4 | 16 | | 109. | public LightVehicle(String tyres, int NoOfTyres, String[] lights, String bodyColor, double discount, boolean enableModify, boolean enableCleaning) { | 15 | 3 | 4 | 7 | 7 | 0 | 21 | 315 | | 110. | super(tyres, NoOfTyres, lights, bodyColor); | 5 | 3 | 0 | 0 | 0 | 0 | 3 | 15 | | 111. | Discount = discount; | 3 | 3 | 0 | 0 | 0 | 0 | 3 | 9 | | 112. | EnableModify = enableModify; | 3 | 3 | 0 | 0 | 0 | 0 | 3 | 9 | | 113. | EnableCleaning = enableCleaning; | 3 | 3 | 0 | 0 | 0 | 0 | 3 | 9 | | 114. | } |  |  |  |  |  |  |  |  | | 115. | public void StartService() { | 2 | 3 | 4 | 0 | 0 | 0 | 7 | 14 | | 116. | System.out.println("Welcome To ABC Service"); | 5 | 3 | 0 | 0 | 0 | 0 | 3 | 15 | | 117. | if(EnableModify) { | 2 | 3 | 0 | 0 | 0 | 2 | 6 | 12 | | 118. | StartModifingVehicle(); | 1 | 3 | 0 | 0 | 0 | 0 | 4 | 4 | | 119. | } |  |  |  |  |  |  |  |  | | 120. | if(EnableCleaning) { | 2 | 3 | 0 | 1 | 0 | 2 | 6 | 12 | | 121. | StartCleaningVehicle(); | 1 | 3 | 0 | 1 | 0 | 0 | 4 | 4 | | 122. | } |  |  |  |  |  |  | 0 | 0 | | 123. | CalculateAndPrintReciept(); | 1 | 3 | 0 | 0 | 0 | 0 | 3 | 3 | | 124. | } |  |  |  |  |  |  |  | 0 | | 125. | public void StartCleaningVehicle() { | 2 | 3 | 4 | 0 | 0 | 0 | 7 | 14 | | 126. | System.out.println(); | 5 | 3 | 0 | 0 | 0 | 0 | 3 | 15 | | 127. | for(int i = 0; i < 10; i++) { | 8 | 3 | 0 | 1 | 0 | 3 | 7 | 56 | | 128. | try { |  |  |  |  |  |  |  | 0 | | 129. | System.out.println(">>>>>>>>>Cleaning Your Vehicle <<<<<<<<<<<<<"); | 6 | 3 | 0 | 0 | 0 | 0 | 3 | 18 | | 130. | Thread.sleep(1000); | 4 | 3 | 0 | 1 | 1 | 0 | 5 | 20 | | 131. | } catch (Exception e) { | 3 | 3 | 0 | 1 | 1 | 0 | 5 | 15 | | 132. | e.printStackTrace(); | 2 | 3 | 0 | 0 | 0 | 0 | 3 | 6 | | 133. | } |  |  |  |  |  |  |  | 0 | | 134. | } |  |  |  |  |  |  |  | 0 | | 135. | System.out.println(); | 5 | 3 | 0 | 0 | 0 | 0 | 3 | 15 | | 136. | System.out.println(">>>>>>>>> Completed Cleaning Task <<<<<<<<<<<<<"); | 6 | 3 | 0 | 0 | 0 | 0 | 3 | 18 | | 137. | } |  |  |  |  |  |  |  | 0 | | 138. | public void StartModifingVehicle() { | 2 | 3 | 4 | 0 | 0 | 0 | 7 | 14 | | 139. | System.out.println(); | 5 | 3 | 0 | 0 | 0 | 0 | 3 | 15 | | 140. | for(int i = 0; i < 10; i++) { | 8 | 3 | 0 | 1 | 0 | 3 | 7 | 56 | | 141. | try { |  |  |  |  |  |  |  | 0 | | 142. | System.out.println(">>>>>>>>>Modifing Your Vehicle <<<<<<<<<<<<<"); | 6 | 3 | 0 | 1 | 0 | 0 | 4 | 24 | | 143. | Thread.sleep(1000); | 4 | 3 | 0 | 1 | 1 | 0 | 5 | 20 | | 144. | } catch (Exception e) { | 3 | 3 | 0 | 1 | 1 | 0 | 5 | 15 | | 145. | e.printStackTrace(); | 2 | 3 | 0 | 0 | 0 | 0 | 3 | 6 | | 146. | } |  |  |  |  |  |  |  | 0 | | 147. | } |  |  |  |  |  |  |  | 0 | | 148. | System.out.println(); | 5 | 3 | 0 | 0 | 0 | 0 | 3 | 15 | | 149. | System.out.println(">>>>>>>>> Completed Modifying Task <<<<<<<<<<<<<"); | 6 | 3 |  | 0 | 0 | 0 | 3 | 18 | | 150. | System.out.println(); | 5 | 3 | 0 | 0 | 0 | 0 | 3 | 15 | | 151. | } |  |  |  |  |  |  |  | 0 | | 152. | public void CalculateAndPrintReciept() { | 2 | 3 | 4 | 0 | 0 | 0 | 7 | 14 | | 153. | if(this.Discount > 0) { | 6 | 3 | 0 | 1 | 0 | 2 | 6 | 36 | | 154. | super.CalculateAndPrintReceipt(Discount); | 5 | 3 | 0 | 1 | 1 | 0 | 5 | 25 | | 155. | }else { |  |  |  |  |  |  |  | 0 | | 156. | super.CalculateAndPrintReceipt(); | 4 | 3 | 0 | 0 | 0 | 0 | 3 | 12 | | 157. | } |  |  |  |  |  |  |  |  | | 158. | } |  |  |  |  |  |  |  |  | | 159. | } |  |  |  |  |  |  |  |  | | 160. | public interface Modify { |  |  |  |  |  |  |  |  | | 161. | public void StartModifingVehicle(); | 2 | 0 | 4 | 0 | 0 | 0 | 4 | 8 | | 162. | } |  |  |  |  |  |  |  |  | | 163. | public interface Cleaning { |  |  |  |  |  |  |  |  | | 164. | public void StartCleaningVehicle(); | 2 | 0 | 4 | 0 | 0 | 0 | 4 | 8 | | 165. | } |  |  |  |  |  |  |  | 0 | | 166. | import java.util.Scanner; |  |  |  |  |  |  |  | 0 | | 167. | public class VehicleMain { |  |  |  |  |  |  |  | 0 | | 168. | public static void main(String[] args) { | 5 | 0 | 4 | 1 | 1 | 0 | 6 | 30 | | 169. | Scanner scn = new Scanner(System.in); | 8 | 0 | 0 | 1 | 1 | 0 | 2 | 16 | | 170. | String[] li = {"BREAKLIGHT","HEADLIGHT"}; | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 171. | System.out.println(">>>>>>>>>Welcome to service Application<<<<<<<<<"); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 172. | System.out.print("Enter Tyre Types : "); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 173. | String Tyre = scn.next(); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 174. | System.out.print("Enter No Of Tyres You Want : "); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 175. | int TyreCount = scn.nextInt(); | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 176. | String[] lights = new String[li.length]; | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 177. | System.out.println("Select Lights "); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 178. | for(int i = 0; i < li.length ; i++) { | 8 | 0 | 0 | 1 | 0 | 3 | 4 | 32 | | 179. | System.out.print("Do you want add new " + li[i] + " . press y to select and press any key to move next : "); | 10 | 0 | 0 | 1 | 0 | 0 | 1 | 10 | | 180. | String res = scn.next(); | 5 | 0 | 0 | 1 | 0 | 0 | 1 | 5 | | 181. | if(res.toUpperCase().equals("y")) { | 7 | 0 | 0 | 2 | 0 | 2 | 4 | 28 | | 182. | lights[i] = li[i]; | 3 | 0 | 0 | 2 | 0 | 0 | 2 | 6 | | 183. | } |  |  |  |  |  |  | 0 | 0 | | 184. | System.out.println(); | 5 | 0 | 0 | 1 | 0 | 0 | 1 | 5 | | 185. | } |  |  |  |  |  |  |  | 0 | | 186. | System.out.print("Enter body color : "); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 187. | String BodyColor = scn.next(); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 188. | System.out.println(); | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 189. | System.out.println(); | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 190. | LightVehicle v1 = new LightVehicle(Tyre, TyreCount, lights, BodyColor, 1500, true, true); | 12 | 0 | 0 | 0 | 6 | 0 | 6 | 72 | | 191. | v1.StartService(); | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 192. | } |  |  |  |  |  |  |  |  | | 193. | } |  |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  |  | 1663 |   **=**  **=28 + 3 + 20 +1- 2**  **= 50**  **Total Number of executable lines = Total number of lines - (Commented lines + White Space + Curly Braces)**  **= 227-(15 +33+42)**  **= 137**  **= (Total Number of executable lines/Total Number of lines) \* 10**  **= (137/227) \* 10**  **= 6.035**  **T \* + + 1663 + 50 + 6.035= 1719.035** |

* Explanation of how the complexity of the first java program was calculated using the newly proposed metric

|  |
| --- |
| **T \* + + 1663 + 50 + 6.035= 1719.035**  Table 1 is showed the tokens and it is used to count the number of tokens in the given program- nT. The token number obtained there has been used in table 2 to get the product of nT \* .  Table 2 is used to calculate the value. is equal to summation of , , and .   * Inheritance Levels including interfaces = * Types of Access Modifiers used in Encapsulation = * Nesting levels of Control Structures = * No of parameters in a method = * Control structures =   Then must be multiplied by the token count. When obtained the product value of each line the total of the product will be obtained at the end of the table as 1663. That is the total product of nT\*  Then we must calculate the value.  =  = Addition of number of inputs and outputs, exceptions, number of functions and number of threads and subtracting the overriding and overloading methods from it.  According to the system the values are as follows:   * 28 * 3 * 20 * 1 * = 2   Then =28 + 3 + 20 +1- 2 and 50 will be obtained as the answer.  Then we must calculate the number of executable lines of the code using the following equation:   * Total number of lines = 227 * Number of commented lines = 15 * Number of white spaces = 33 * Number of curly braces = 42   Total Number of executable lines = Total number of lines - (Commented lines + White Space + Curly Braces)  = 227-(15 +33+42)  = 137  = (Total Number of executable lines/Total Number of lines) \* 10 = (137/227) \* 10 = 6.035  Then when we substitute all of these values into the metrics equation, we get the final output as follows:  **T \* + + 1663 + 50 + 6.035= 1719.035** |

* Calculation of complexity of the second java program using the newly proposed metric

|  |
| --- |
| Source code of the second java program  public abstract class Employee {  private String EmpID;  private String Name;  private String NIC;  private String MobileNumber;    public Employee(String empID, String name, String nIC, String mobileNumber) {  EmpID = empID;  Name = name;  NIC = nIC;  MobileNumber = mobileNumber;  }  public String getEmpID() {  return EmpID;  }  public void setEmpID(String empID) {  EmpID = empID;  }  public String getName() {  return Name;  }  public void setName(String name) {  Name = name;  }  public String getNIC() {  return NIC;  }  public void setNIC(String nIC) {  NIC = nIC;  }  public String getMobileNumber() {  return MobileNumber;  }  public void setMobileNumber(String mobileNumber) {  MobileNumber = mobileNumber;  }    public abstract void CalculateSalary();    public abstract void CalculateSalary(double bonus);  }  public class FullTimeEmployee extends Employee implements SalaryIncements,Promotions{  private double Bonus;  private int WorkedDays;  private boolean Status;    public FullTimeEmployee(String empID, String name, String nIC, String mobileNumber, int workedDays) {  super(empID, name, nIC, mobileNumber);  WorkedDays = workedDays;  this.Status = false;  this.Bonus = 0;  }  //For Controll all methods in Full time Employee class  public void Activate() {  Addincrements();  System.out.println();  CheckPromotions();  System.out.println();  if(Bonus > 0) {  CalculateSalary(this.Bonus);  }else {  CalculateSalary();  }  }    //for calculate salary without bonus  @Override  public void CalculateSalary() {  System.out.println(">>>>>>>>Salary Calculation is running<<<<<<<<");  for(int i = 0; i < 10; i++) {  try {  Thread.sleep(1000);  System.out.println("Calculating...........");    } catch (Exception e) {  e.printStackTrace();  }  }    System.out.println();  System.out.println("Total Salary : " + 15000);  }  //for calculate salary with bonus  @Override  public void CalculateSalary(double bonus) {  System.out.println(">>>>>>>>Salary Calculation is running<<<<<<<<");  for(int i = 0; i < 10; i++) {  try {  Thread.sleep(1000);  System.out.println("Calculating...........");  } catch (Exception e) {  e.printStackTrace();  }  }  System.out.println();  System.out.println("Total Salary : " + (15000 + bonus));    }  //for check bonus  @Override  public void CheckPromotions() {  if(this.Status) {  System.out.println("Congradulations!!!");  }else {  System.out.println("Pending!!!");  }  }  //for add increments  @Override  public void Addincrements() {  if(WorkedDays > 20) {  System.out.println("Congradulations.. You have a bonus");  Bonus = 5000;  Status =true;  }else {  System.out.println("Congradulations.. You have ");  Bonus = 5000;  }    }      }  public interface SalaryIncements {  public void Addincrements();  }  public interface Promotions {  public void CheckPromotions();  }  import java.util.Scanner;  public class TestMMain {  public static void main(String[] args) {  Scanner scn = new Scanner(System.in);  System.out.println(">>>>>>>Fill your Details<<<<<<<");    System.out.print("Enter Your Name : ");  String name = scn.next();    System.out.print("Enter Your Empolyee Id : ");  String empId = scn.next();    System.out.print("Enter Your NIC : ");  String nic = scn.next();    System.out.print("Enter Your Mobile Number : ");  String mobile = scn.next();    System.out.print("Enter Your Worked Days : ");  int days = scn.nextInt();    FullTimeEmployee e1 = new FullTimeEmployee(empId, name, nic, mobile, days);  e1.Activate();  }  } |
| Calculation of complexity of the second java program  Calculating the Token Count -**nT**  Table 1   |  |  |  |  | | --- | --- | --- | --- | | **Line No** | **Program Statements** | **Tokens** | **nT(Token Count)** | | 1. | public abstract class Employee { |  |  | | 2. | private String EmpID; | String | 1 | | 3. | private String Name; | String | 1 | | 4. | private String NIC; | String | 1 | | 5. | private String MobileNumber; | String | 1 | | 6. | public Employee(String empID, String name, String nIC, String mobileNumber) { | Employee() | 1 | | 7. | EmpID = empID; | EmpID, = , empID | 3 | | 8. | Name = name; | Name, = , name | 3 | | 9. | NIC = nIC; | NIC ,= ,nIC | 3 | | 10. | MobileNumber = mobileNumber; | MobileNumber, =, mobileNumber | 3 | | 11. | } |  |  | | 12. | public String getEmpID() { | String, getEmpID() | 2 | | 13. | return EmpID; | EmpID | 1 | | 14. | } |  | 0 | | 15. | public void setEmpID(String empID) { | Void, setEmpID() | 2 | | 16. | EmpID = empID; | EmpID, = , empID | 3 | | 17. | } |  |  | | 18. | public String getName() { | String, getName() | 2 | | 19. | return Name; | Name | 1 | | 20. | } |  |  | | 21. | public void setName(String name) { | void, setName() | 2 | | 22. | Name = name; | Name, = , name | 3 | | 23. | } |  |  | | 24. | public String getNIC() { | String, getNIC() | 2 | | 25. | return NIC; | NIC | 1 | | 26. | } |  |  | | 27. | public void setNIC(String nIC) { | void, setNIC() | 2 | | 28. | NIC = nIC; | NIC ,= ,nIC | 3 | | 29. | } |  |  | | 30. | public String getMobileNumber() { | String, getMobileNumber() | 2 | | 31. | return MobileNumber; | MobileNumber | 1 | | 32. | } |  |  | | 33. | public void setMobileNumber(String mobileNumber) { | void, setMobileNumber () | 2 | | 34. | MobileNumber = mobileNumber; | MobileNumber, =, mobileNumber | 3 | | 35. | } |  |  | | 36. | public abstract void CalculateSalary(); | Abstract, void, CalculateSalary() | 3 | | 37. | public abstract void CalculateSalary(double bonus); | Abstract, void, CalculateSalary(); | 3 | | 38. | } |  |  | | 39. | public class FullTimeEmployee extends Employee implements SalaryIncements,Promotions { |  |  | | 40. | private double Bonus; | double | 1 | | 41. | private int WorkedDays; | Int | 1 | | 42. | private boolean Status; | Boolean | 1 | | 43. | public FullTimeEmployee(String empID, String name, String nIC, String mobileNumber, int workedDays) { | FullTimeEmployee() | 1 | | 44. | super(empID, name, nIC, mobileNumber); | super(),empID, name, nIC, mobileNumber | 5 | | 45. | WorkedDays = workedDays; | WorkedDays, =, workedDays | 3 | | 46. | this.Status = false; | this, . , Status, =, false | 5 | | 47. | this.Bonus = 0; | this, . , Bonus, =, 0 | 5 | | 48. | } |  |  | | 49. | //For Controll all methods in Full time Employee class |  |  | | 50. | public void Activate() { | void, Activate() | 2 | | 51. | Addincrements(); | Addincrements() | 1 | | 52. | System.out.println(); | System , . , out, . ,println() | 5 | | 53. | CheckPromotions(); | CheckPromotions() | 1 | | 54. | System.out.println(); | System , . , out, . ,println() | 5 | | 55. | if(Bonus > 0) { | if(),Bonus, >, 0 | 4 | | 56. | CalculateSalary(this.Bonus); | CalculateSalary() | 1 | | 57. | }else { |  |  | | 58. | CalculateSalary(); | CalculateSalary() | 1 | | 59. | } |  |  | | 60. | } |  |  | | 61. | //for calculate salary without bonus |  |  | | 62. | @Override |  |  | | 63. | public void CalculateSalary() { | void ,CalculateSalary() | 2 | | 64. | System.out.println(">>>>>>>>Salary Calculation is running<<<<<<<<"); | System, . , out, . , println(),">>>>>>>>Salary Calculation is running<<<<<<<<” | 6 | | 65. | for(int i = 0; i < 10; i++) { | for() , int, i, =, 0, i, <, 10, i, ++ | 10 | | 66. | try { |  |  | | 67. | Thread.sleep(1000); | Thread, . , sleep(), 1000 | 4 | | 68. | System.out.println("Calculating..........."); | System, . , out, . , println(),"Calculating..........." | 6 | | 69. | } catch (Exception e) { | catch(), Exception, e | 3 | | 70. | e.printStackTrace(); | e, . , printStackTrace() | 2 | | 71. | } |  |  | | 72. | } |  |  | | 73. | System.out.println(); | System, . , out, . , println() | 5 | | 74. | System.out.println("Total Salary : " + 15000); | System, . , out, . , println(),"Total Salary : ", +, 15000 | 8 | | 75. | } |  |  | | 76. | //for calculate salary with bonus |  |  | | 77. | @Override |  |  | | 78. | public void CalculateSalary(double bonus) { | void, CalculateSalary () | 2 | | 79. | System.out.println(">>>>>>>>Salary Calculation is running<<<<<<<<"); | System, . , out, . , println(),">>>>>>>>Salary Calculation is running<<<<<<<<" | 6 | | 80. | for(int i = 0; i < 10; i++) { | for() , int, i, =, 0, i, <, 10, i, ++ | 10 | | 81. | Thread.sleep(1000); | Thread, . , sleep(), 1000 | 4 | | 82. | System.out.println("Calculating..........."); | System, . , out, . , println(),"Calculating..........." | 6 | | 83. | } catch (Exception e) { | catch(), Exception, e | 3 | | 84. | e.printStackTrace(); | e, ., printStackTrace() | 3 | | 85. | } |  |  | | 86. | } |  |  | | 87. | System.out.println(); | System, . , out, . , println() | 5 | | 88. | System.out.println("Total Salary : " + (15000 + bonus)); | System, . , out, . , println(),"Total Salary : ", +, 15000, + , bonus | 10 | | 89. | } |  |  | | 90. | //for check bonus |  |  | | 91. | @Override |  |  | | 92. | public void CheckPromotions() { | Void, CheckPromotions() | 2 | | 93. | if(this.Status) { | if(),this, . , Status | 4 | | 94. | System.out.println("Congradulations!!!"); | System, . , out, . , println(), “Congradulations!!!” | 6 | | 95. | }else { |  |  | | 96. | System.out.println("Pending!!!"); | System, . , out, . , println(), “Pending!!!” | 6 | | 97. | } |  |  | | 98. | } |  |  | | 99. | //for add increments |  |  | | 100. | @Override |  |  | | 101. | public void Addincrements() { | void , Addincrements() | 2 | | 102. | if(WorkedDays > 20) { | if(),WorkedDays , > , 20 | 4 | | 103. | System.out.println("Congradulations.. You have a bonus"); | System, . , out, . , println(),"Congradulations.. You have a bonus" | 6 | | 104. | Bonus = 5000; | Bonus, =, 5000 | 3 | | 105. | Status =true; | Status , =, true | 3 | | 106. | }else { |  |  | | 107. | System.out.println("Congradulations.. You have "); | System, . , out, . , println(),"Congradulations.. You have" | 6 | | 108. | Bonus = 5000; | Bonus, =, 5000 | 3 | | 109. | } |  |  | | 110. | } |  |  | | 111. | } |  |  | | 112. | public interface SalaryIncements { |  |  | | 113. | public void Addincrements(); | void, Addincrements() | 2 | | 114. | } |  |  | | 115. | public interface Promotions { |  |  | | 116. | public void CheckPromotions(); | void, CheckPromotions(); | 2 | | 117. | } |  |  | | 118. | import java.util.Scanner; |  |  | | 119. | public class TestMMain { |  |  | | 120. | public static void main(String[] args) { | static, void, main(), String[], args | 5 | | 121. | Scanner scn = new Scanner(System.in); | Scanner, scn, = , new, Scanner(), System, ., in | 8 | | 122. | System.out.println(">>>>>>>Fill your Details<<<<<<<"); | System, . , out, . , println(),">>>>>>>Fill your Details<<<<<<<" | 6 | | 123. | System.out.print("Enter Your Name : "); | System, ., out, . , println(),"Enter Your Name : " | 6 | | 124. | String name = scn.next(); | String , name, = , scn, . , next() | 6 | | 125. | System.out.print("Enter Your Empolyee Id : "); | System, . , out, . , println(),"Enter Your Empolyee Id : " | 6 | | 126. | String empId = scn.next(); | String , empId, = , scn, . , next() | 6 | | 127. | System.out.print("Enter Your NIC : "); | System, ., out, . , println(),"Enter Your NIC : " | 6 | | 128. | String nic = scn.next(); | String , nic, = , scn, . , next(); | 6 | | 129. | System.out.print("Enter Your Mobile Number : "); | System, ., out, . , println(),"Enter Your Mobile Number : " | 6 | | 130. | String mobile = scn.next(); | String , mobile, = , scn, . , next(); | 6 | | 131. | System.out.print("Enter Your Worked Days : "); | System, ., out, . , println(), "Enter Your Worked Days : " | 6 | | 132. | int days = scn.nextInt(); | int , days, = , scn, . , nextInt(); | 6 | | 133. | FullTimeEmployee e1 = new FullTimeEmployee(empId, name, nic, mobile, days); | FullTimeEmployee, e1, =, new , FullTimeEmployee() | 5 | | 134. | e1.Activate(); | e1 , . , Activate() | 3 | | 135. | } |  |  | | 136. | } |  |  |   Calculation of complexity of the first java program  Calculating nT \*  Table 2   |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Line No** | **Program Statements** | **nT(Token Count)** |  |  |  |  |  |  | **nT\*** | | 1. | public abstract class Employee { |  |  |  |  |  |  |  |  | | 2. | private String EmpID; | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | | 3. | private String Name; | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | | 4. | private String NIC; | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | | 5. | private String MobileNumber; | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | | 6. | public Employee(String empID, String name, String nIC, String mobileNumber) { | 1 | 0 | 4 | 0 | 4 | 0 | 8 | 8 | | 7. | EmpID = empID; | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 8. | Name = name; | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 9. | NIC = nIC; | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 10. | MobileNumber = mobileNumber; | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 11. | } |  |  |  |  |  |  |  |  | | 12. | public String getEmpID() { | 2 | 0 | 4 | 0 | 0 | 0 | 4 | 8 | | 13. | return EmpID; | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 14. | } |  |  |  |  |  |  |  |  | | 15. | public void setEmpID(String empID) { | 2 | 0 | 4 | 0 | 1 | 0 | 5 | 10 | | 16. | EmpID = empID; | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 17. | } |  |  |  |  |  |  |  |  | | 18. | public String getName() { | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 19. | return Name; | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 20. | } |  |  |  |  |  |  |  |  | | 21. | public void setName(String name) { | 2 | 0 | 4 | 0 | 1 | 0 | 5 | 10 | | 22. | Name = name; | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 23. | } |  |  |  |  |  |  |  |  | | 24. | public String getNIC() { | 2 | 0 | 4 | 0 | 0 | 0 | 4 | 8 | | 25. | return NIC; | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 26. | } |  |  |  |  |  |  |  |  | | 27. | public void setNIC(String nIC) { | 2 | 0 | 4 | 0 | 1 | 0 | 5 | 10 | | 28. | NIC = nIC; | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 29. | } |  |  |  |  |  |  |  |  | | 30. | public String getMobileNumber() { | 2 | 0 | 4 | 0 | 0 | 0 | 4 | 8 | | 31. | return MobileNumber; | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 32. | } |  |  |  |  |  |  |  |  | | 33. | public void setMobileNumber(String mobileNumber) { | 2 | 0 | 4 | 0 | 1 | 0 | 5 | 10 | | 34. | MobileNumber = mobileNumber; | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 35. | } |  |  |  |  |  |  |  |  | | 36. | public abstract void CalculateSalary(); | 3 | 0 | 4 | 0 | 0 | 0 | 4 | 12 | | 37. | public abstract void CalculateSalary(double bonus); | 3 | 0 | 4 | 0 | 1 | 0 | 5 | 15 | | 38. | } |  |  |  |  |  |  |  |  | | 39. | public class FullTimeEmployee extends Employee implements SalaryIncements,Promotions { |  |  |  |  |  |  |  |  | | 40. | private double Bonus; | 1 | 3 | 1 | 0 | 0 | 0 | 4 | 4 | | 41. | private int WorkedDays; | 1 | 3 | 1 | 0 | 0 | 0 | 4 | 4 | | 42. | private boolean Status; | 1 | 3 | 1 | 0 | 0 | 0 | 4 | 4 | | 43. | public FullTimeEmployee(String empID, String name, String nIC, String mobileNumber, int workedDays) { | 1 | 3 | 4 | 0 | 5 | 0 | 12 | 12 | | 44. | super(empID, name, nIC, mobileNumber); | 5 | 3 | 0 | 0 | 4 | 0 | 7 | 35 | | 45. | WorkedDays = workedDays; | 3 | 3 | 0 | 0 | 0 | 0 | 3 | 9 | | 46. | this.Status = false; | 5 | 3 | 0 | 0 | 0 | 0 | 3 | 15 | | 47. | this.Bonus = 0; | 5 | 3 | 0 | 0 | 0 | 0 | 3 | 15 | | 48. | } |  |  |  |  |  |  |  |  | | 49. | //For Controll all methods in Full time Employee class |  |  |  |  |  |  |  |  | | 50. | public void Activate() { | 2 | 3 | 4 | 0 | 0 | 0 | 7 | 14 | | 51. | Addincrements(); | 1 | 3 | 0 | 0 | 0 | 0 | 3 | 3 | | 52. | System.out.println(); | 5 | 3 | 0 | 0 | 0 | 0 | 3 | 15 | | 53. | CheckPromotions(); | 1 | 3 | 0 | 0 | 0 | 0 | 3 | 3 | | 54. | System.out.println(); | 5 | 3 | 0 | 0 | 0 | 0 | 3 | 15 | | 55. | if(Bonus > 0) { | 4 | 3 | 0 | 1 | 0 | 2 | 6 | 24 | | 56. | CalculateSalary(this.Bonus); | 1 | 3 | 0 | 1 | 0 | 0 | 4 | 4 | | 57. | }else { |  |  |  |  |  |  |  |  | | 58. | CalculateSalary(); | 1 | 3 | 0 | 1 | 0 | 0 | 4 | 4 | | 59. | } |  |  |  |  |  |  |  |  | | 60. | } |  |  |  |  |  |  |  |  | | 61. | //for calculate salary without bonus |  |  |  |  |  |  |  |  | | 62. | @Override |  |  |  |  |  |  |  |  | | 63. | public void CalculateSalary() { | 2 | 3 | 4 | 0 | 0 | 0 | 7 | 14 | | 64. | System.out.println(">>>>>>>>Salary Calculation is running<<<<<<<<"); | 6 | 3 | 0 | 0 | 0 | 0 | 3 | 18 | | 65. | for(int i = 0; i < 10; i++) { | 10 | 3 | 0 | 1 | 0 | 3 | 7 | 70 | | 66. | try { |  |  |  |  |  |  |  |  | | 67. | Thread.sleep(1000); | 4 | 3 | 0 | 1 | 1 | 0 | 5 | 20 | | 68. | System.out.println("Calculating..........."); | 6 | 3 | 0 | 1 | 0 | 0 | 4 | 24 | | 69. | } catch (Exception e) { | 3 | 3 | 0 | 1 | 1 | 0 | 5 | 15 | | 70. | e.printStackTrace(); | 2 | 3 | 0 | 1 | 0 | 0 | 4 | 8 | | 71. | } |  |  |  |  |  |  |  |  | | 72. | } |  |  |  |  |  |  |  |  | | 73. | System.out.println(); | 5 | 3 | 0 | 0 | 0 | 0 | 3 | 15 | | 74. | System.out.println("Total Salary : " + 15000); | 8 | 3 | 0 | 0 | 0 | 0 | 3 | 24 | | 75. | } |  |  |  |  |  |  |  |  | | 76. | //for calculate salary with bonus |  |  |  |  |  |  |  |  | | 77. | @Override |  |  |  |  |  |  |  |  | | 78. | public void CalculateSalary(double bonus) { | 2 | 3 | 4 | 0 | 1 | 0 | 8 | 16 | | 79. | System.out.println(">>>>>>>>Salary Calculation is running<<<<<<<<"); | 6 | 3 | 0 | 0 | 0 | 0 | 3 | 18 | | 80. | for(int i = 0; i < 10; i++) { | 10 | 3 | 0 | 1 | 0 | 3 | 7 | 70 | | 81. | Thread.sleep(1000); | 4 | 3 | 0 | 1 | 1 | 0 | 5 | 20 | | 82. | System.out.println("Calculating..........."); | 6 | 3 | 0 | 1 | 0 | 0 | 4 | 24 | | 83. | } catch (Exception e) { | 3 | 3 | 0 | 1 | 1 | 0 | 5 | 15 | | 84. | e.printStackTrace(); | 3 | 3 | 0 | 1 | 0 | 0 | 4 | 12 | | 85. | } |  |  |  |  |  |  |  |  | | 86. | } |  |  |  |  |  |  |  |  | | 87. | System.out.println(); | 5 | 3 | 0 | 0 | 0 | 0 | 3 | 15 | | 88. | System.out.println("Total Salary : " + (15000 + bonus)); | 10 | 3 | 0 | 0 | 0 | 0 | 3 | 30 | | 89. | } |  |  |  |  |  |  |  |  | | 90. | //for check bonus |  |  |  |  |  |  |  |  | | 91. | @Override |  |  |  |  |  |  |  |  | | 92. | public void CheckPromotions() { | 2 | 3 | 4 | 0 | 0 | 0 | 7 | 14 | | 93. | if(this.Status) { | 4 | 3 | 0 | 1 | 0 | 2 | 6 | 24 | | 94. | System.out.println("Congradulations!!!"); | 6 | 3 | 0 | 1 | 0 | 0 | 4 | 24 | | 95. | }else { |  |  |  |  |  |  |  |  | | 96. | System.out.println("Pending!!!"); | 6 | 3 | 0 | 1 | 0 | 0 | 4 | 24 | | 97. | } |  |  |  |  |  |  |  |  | | 98. | } |  |  |  |  |  |  |  |  | | 99. | //for add increments |  |  |  |  |  |  |  |  | | 100. | @Override |  |  |  |  |  |  |  |  | | 101. | public void Addincrements() { | 2 | 3 | 4 | 0 | 0 | 0 | 7 | 14 | | 102. | if(WorkedDays > 20) { | 4 | 3 | 0 | 1 | 0 | 2 | 6 | 24 | | 103. | System.out.println("Congradulations.. You have a bonus"); | 6 | 3 | 0 | 1 | 0 | 0 | 4 | 24 | | 104. | Bonus = 5000; | 3 | 3 | 0 | 1 | 0 | 0 | 4 | 12 | | 105. | Status =true; | 3 | 3 | 0 | 1 | 0 | 0 | 4 | 12 | | 106. | }else { |  |  |  |  |  |  |  |  | | 107. | System.out.println("Congradulations.. You have "); | 6 | 3 | 0 | 1 | 0 | 0 | 4 | 24 | | 108. | Bonus = 5000; | 3 | 3 | 0 | 1 | 0 | 0 | 4 | 12 | | 109. | } |  |  |  |  |  |  |  |  | | 110. | } |  |  |  |  |  |  |  |  | | 111. | } |  |  |  |  |  |  |  |  | | 112. | public interface SalaryIncements { |  |  |  |  |  |  |  |  | | 113. | public void Addincrements(); | 2 | 0 | 4 | 0 | 0 | 0 | 4 | 8 | | 114. | } |  |  |  |  |  |  |  |  | | 115. | public interface Promotions { |  |  |  |  |  |  |  |  | | 116. | public void CheckPromotions(); | 2 | 0 | 4 | 0 | 0 | 0 | 4 | 8 | | 117. | } |  |  |  |  |  |  |  |  | | 118. | import java.util.Scanner; |  |  |  |  |  |  |  |  | | 119. | public class TestMMain { |  |  |  |  |  |  |  |  | | 120. | public static void main(String[] args) { | 5 | 0 | 4 | 0 | 1 | 0 | 5 | 25 | | 121. | Scanner scn = new Scanner(System.in); | 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 122. | System.out.println(">>>>>>>Fill your Details<<<<<<<"); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 123. | System.out.print("Enter Your Name : "); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 124. | String name = scn.next(); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 125. | System.out.print("Enter Your Empolyee Id : "); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 126. | String empId = scn.next(); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 127. | System.out.print("Enter Your NIC : "); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 128. | String nic = scn.next(); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 129. | System.out.print("Enter Your Mobile Number : "); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 130. | String mobile = scn.next(); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 131. | System.out.print("Enter Your Worked Days : "); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 132. | int days = scn.nextInt(); | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 133. | FullTimeEmployee e1 = new FullTimeEmployee(empId, name, nic, mobile, days); | 5 | 0 | 0 | 0 | 5 | 0 | 5 | 25 | | 134. | e1.Activate(); | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 135. | } |  |  |  |  |  |  |  |  | | 136. | } |  |  |  |  |  |  |  |  | | 137. |  | | | | | | | | 985 |   **=**  **=25 + 2 + 14 +1- 4**  **= 38**  **Total Number of executable lines = Total number of lines - (Commented lines + White Space + Curly Braces)**  **= 175-(5 +35+29)**  **= 106**  **= (Total Number of executable lines/Total Number of lines) \* 10**  **= (106/175 ) \* 10**  **= 6.05**  **T \* + + + 38 + 6.05= 1029.05** |

* Explanation of how the complexity of the second java program was calculated using the newly proposed metric

|  |
| --- |
| **T \* + + + 38 + 6.05= 1029.05**  Table 1 is showed the tokens and it is used to count the number of tokens in the given program- nT. The token number obtained there has been used in table 2 to get the product of nT \* .  Table 2 is used to calculate the value. is equal to summation of , , and .   * Inheritance Levels including interfaces = * Types of Access Modifiers used in Encapsulation = * Nesting levels of Control Structures = * No of parameters in a method = * Control structures =   Then must be multiplied by the token count. When obtained the product value of each line the total of the product will be obtained at the end of the table as 985. That is the total product of nT\*  Then we must calculate the value.  =  = Addition of number of inputs and outputs, exceptions, number of functions and number of threads and subtracting the overriding and overloading methods from it.  According to the system the values are as follows:   * =   Then =25 + 2 + 14 +1- 4 and 38 will be obtained as the answer.  Then we must calculate the number of executable lines of the code using the following equation:   * Total number of lines = **175** * Number of commented lines = **5** * Number of white spaces = **35** * Number of curly braces = **29**   Total Number of executable lines = Total number of lines - (Commented lines + White Space + Curly Braces)  = 175-(5 +35+29)  = 106  = (Total Number of executable lines/Total Number of lines) \* 10 = (106/175) \* 10 = 6.05  Then when we substitute all these values into the metrics equation, we get the final output as follows:  **T \* + + + 38 + 6.05= 1029.05** |

* References

|  |  |  |
| --- | --- | --- |
| [1] Misra, Sanjay, and Ibrahim Akman. "Weighted class complexity: a measure of complexity for object oriented system." Journal of Information Science and Engineering 24 (2008): 1689-1708.  [2] Chhillar, Usha, and Sucheta Bhasin. "A new weighted composite complexity measure for object-oriented systems." International journal of information and communication technology research 1, no. 3 (2011).   |  |  | | --- | --- | | [3] | A. A. L. F.-S. A. R. D. SANJAY MISRA, "A Suite of Object Oriented Cognitive Complexity Metrics," IEEE Access, 2018. |   [4] McCabe(1976) : Complexity Measure, IEEE Transactions on Software Engineering, Volume 2, No 4, pp 308-320, December 1976  [5] Bhatia, Sonam, and Jyoteesh Malhotra. "A survey on impact of lines of code on software complexity." In 2014 International Conference on Advances in Engineering & Technology Research (ICAETR-2014), pp. 1-4. IEEE, 2014. |