**BAKER’S CAKE**

**PROJECT FILE**

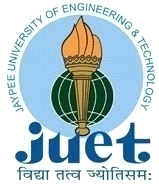
*Submitted by*

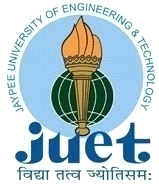
**Vivek Gupta (181B246)**

**Pyush Kumar Gupta (181B261)**

Under the guidance of: **Dr. Prateek Pandey**

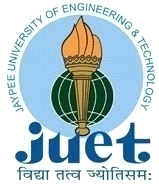
**Dr. Dinesh Kumar Verma**

****

****

**PROBLEM STATEMENT**

The system that our project code implements is based out of a bakery cake baking procedure. A customer went to a bakery and places the order as per his/her requirement for the cake. He has options for the base for the cake, that is, Red-Velvet Cake and Chocolate Cake. After selecting the base ingredient for the cake, the cake base undergoes the baking procedure. As this procedure finishes the customer is asked about the decoration for the cake. The options for the decorations are Choco-Chips, Strawberries and Gems. After providing the options the second procedure for the decoration starts and as it finishes the customer gets the final output Cake and the final price.



**DESIGN PATTERNS**

The design patterns used in our project are:-

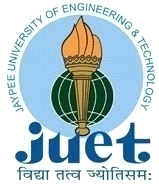
1. Template Method Design Pattern
2. Decorator Pattern

A Template Pattern says that **"just define the skeleton of a function in an operation, deferring some steps to its subclasses".** The advantage of using this pattern is that it provides code reusability. It is used when the common behavior among sub-classes should be moved to a single common class by avoiding the duplication.

The Template Method Design Pattern is used for the creation of the Cake’s base. The class Bakery contains the template method which has the fixed template as:-

* basicPreparations(): void
* buildBatter(): void
* buildBake(): void
* buildLayers(): void
* buildIcing(): void

This method gives the step by step procedure of preparing the Cake’s Base.



The second design pattern used is Decorator Design Pattern. A Decorator Pattern says that just **"attach a flexible additional responsibilities to an object dynamically".** In other words, The Decorator Pattern uses composition instead of inheritance to extend the functionality of an object at runtime. The advantage of this design pattern is that it provides greater flexibility than static inheritance.

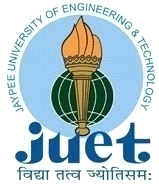
In our project this design pattern is used to provide the extended functionality of the cake’s decoration. The CakeDecorator class extends the Cake interface containing the function as prepareCake() and cakePrice(). The decoration class is extended by the following decorating classed:-

* Gems
* ChocoChips
* Strawberries

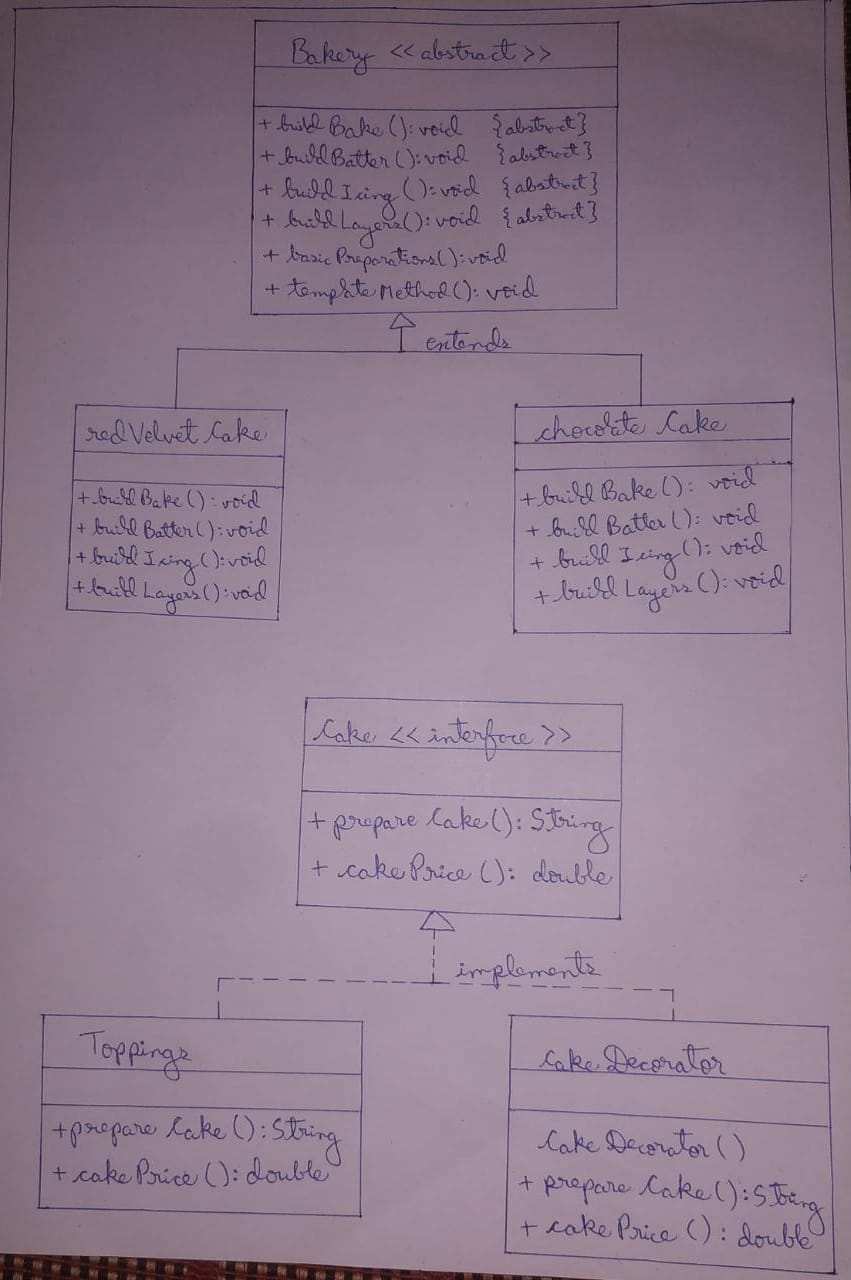
Each of these classes overrides the following functions:-

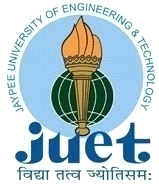
* prepareCake(): String
* cakePrice(): double

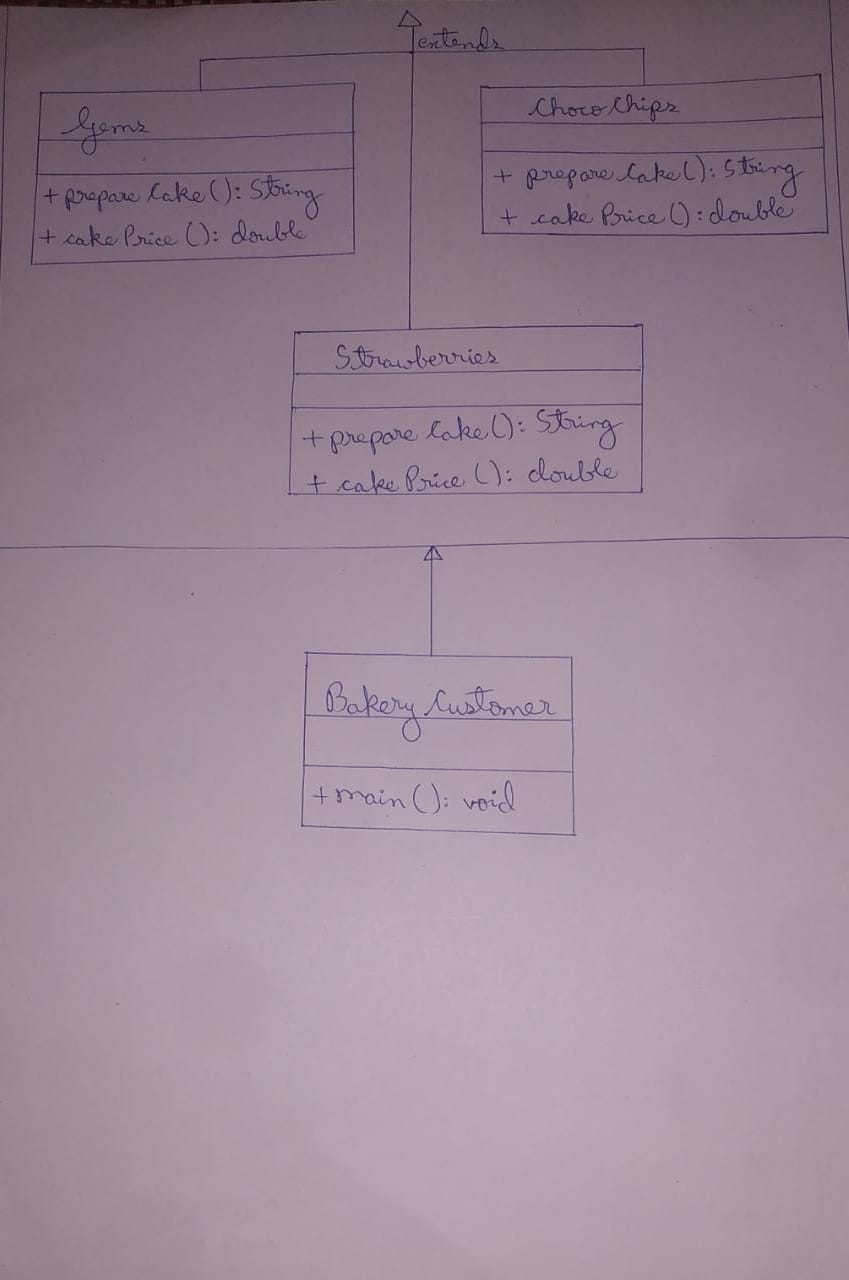
These functions decorates the cake based on the option chosen and provides the output with the total price, that is, the sum of the base price, which is Rs. 500 and the addition price based in the decoration decided.

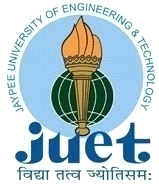


**CLASS DIAGRAM**

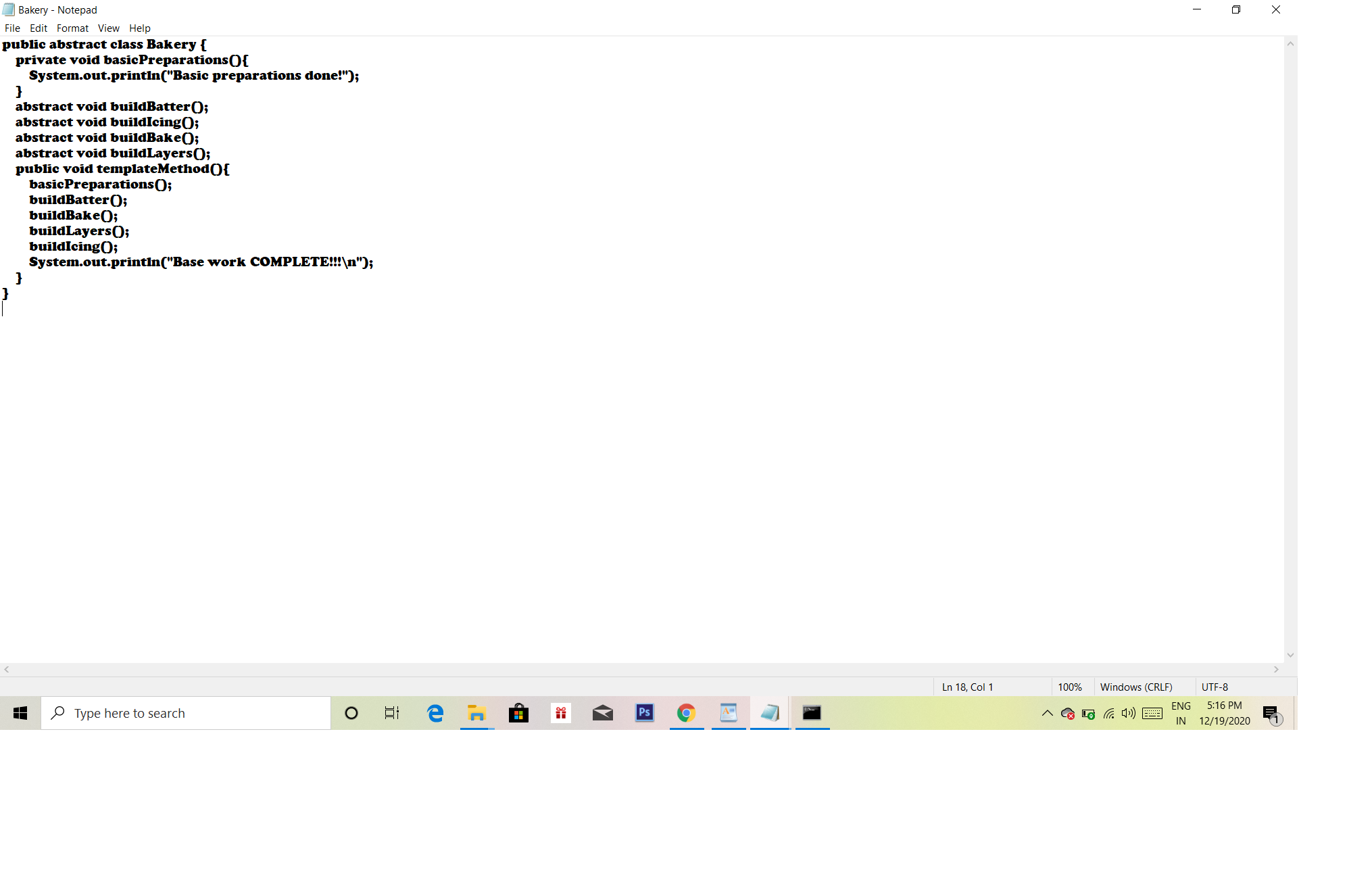
****

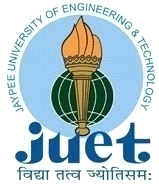


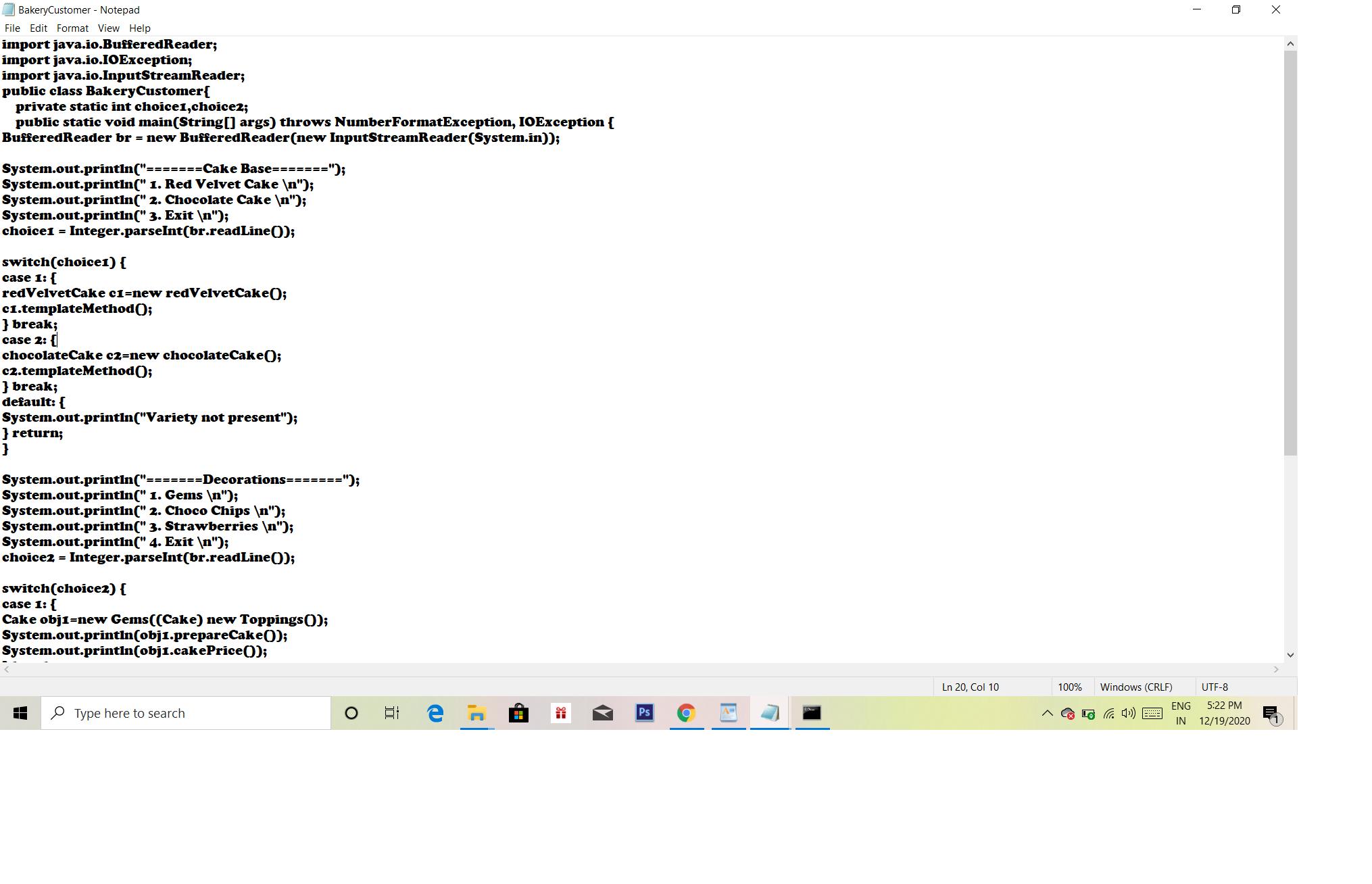


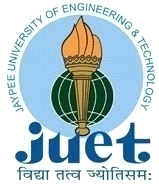


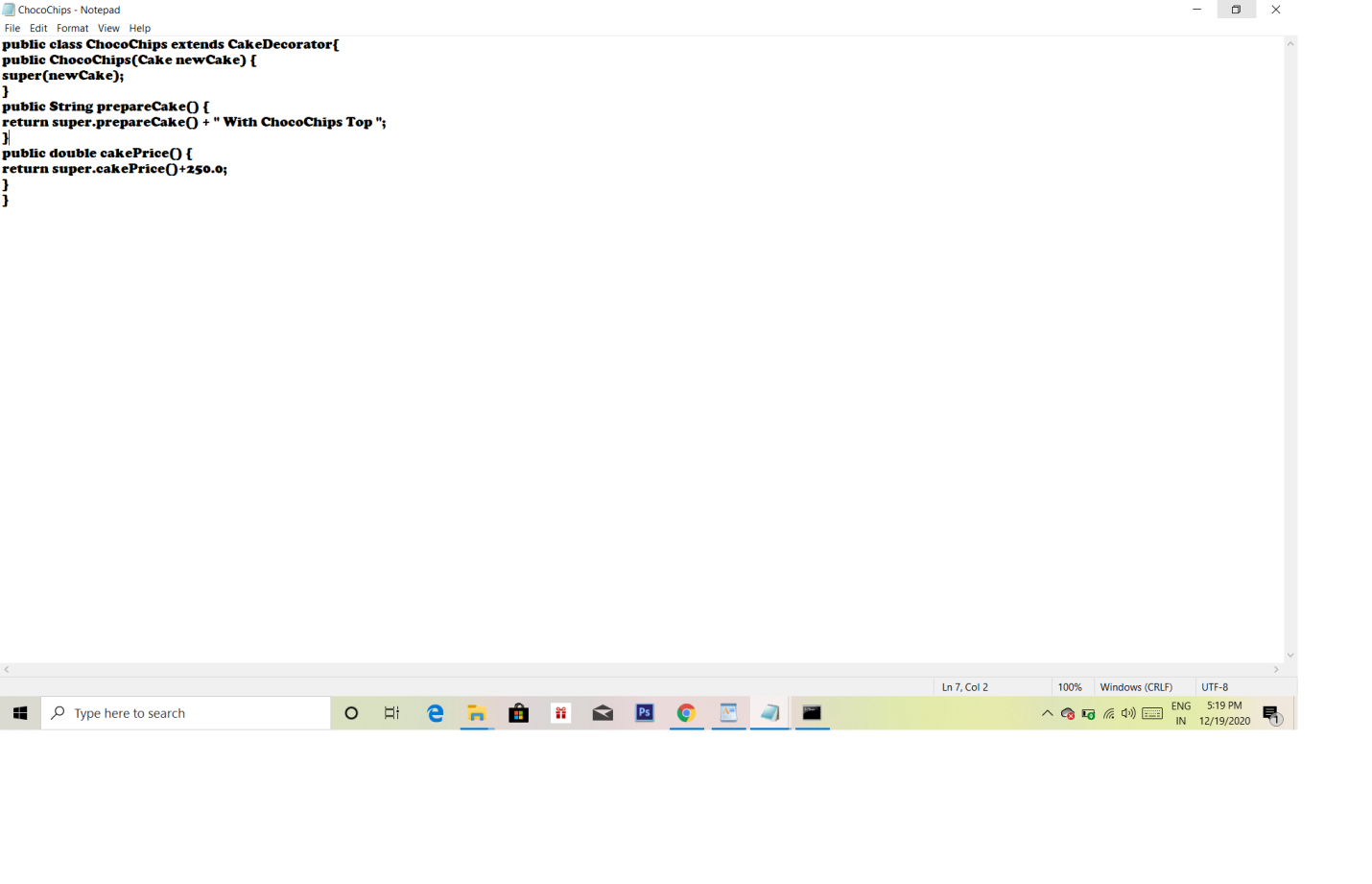
**CODE SCREENSHOTS**

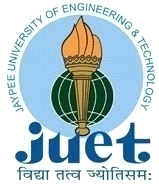
****

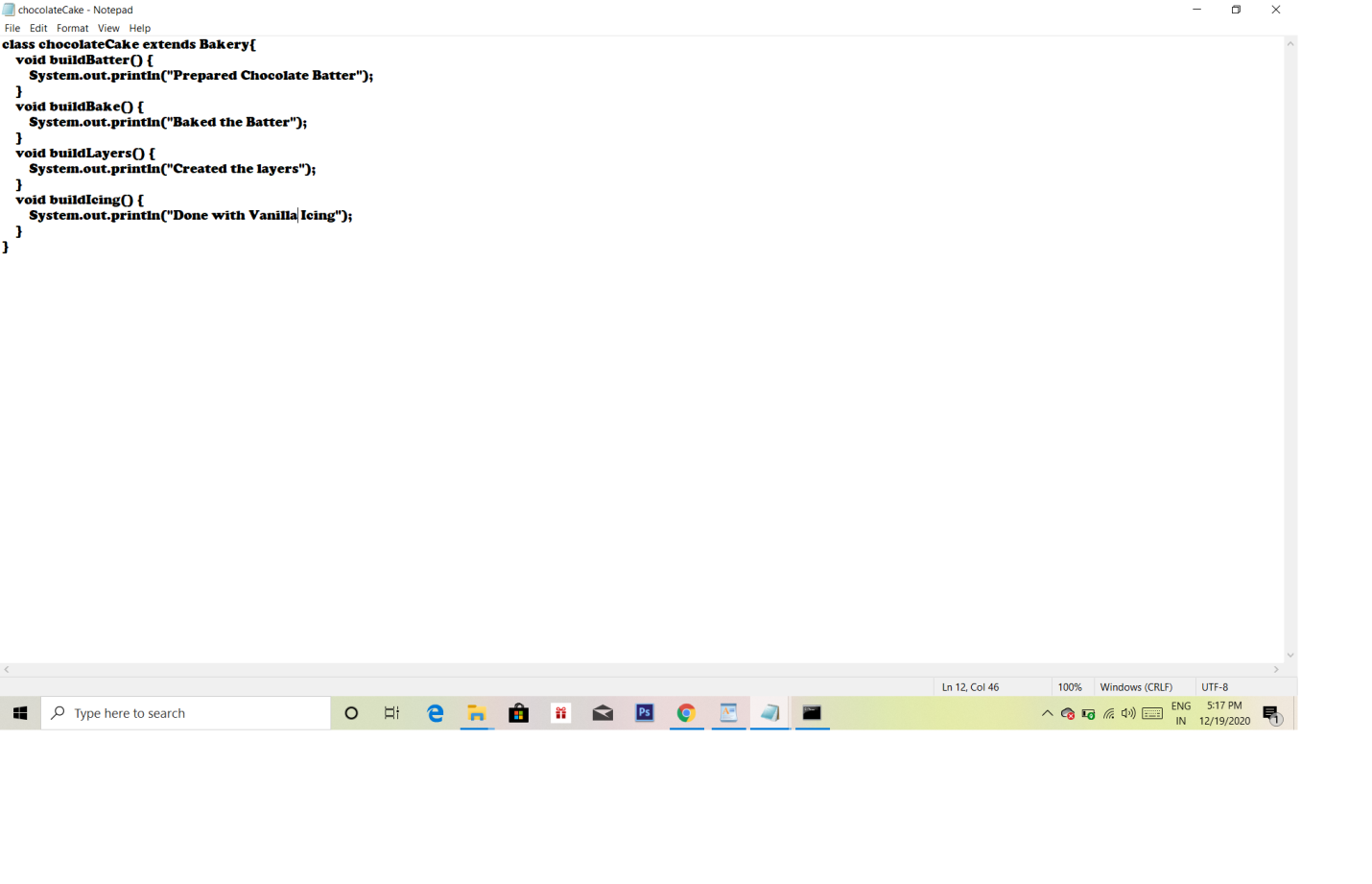
****

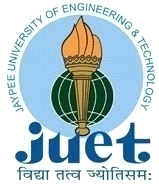
****

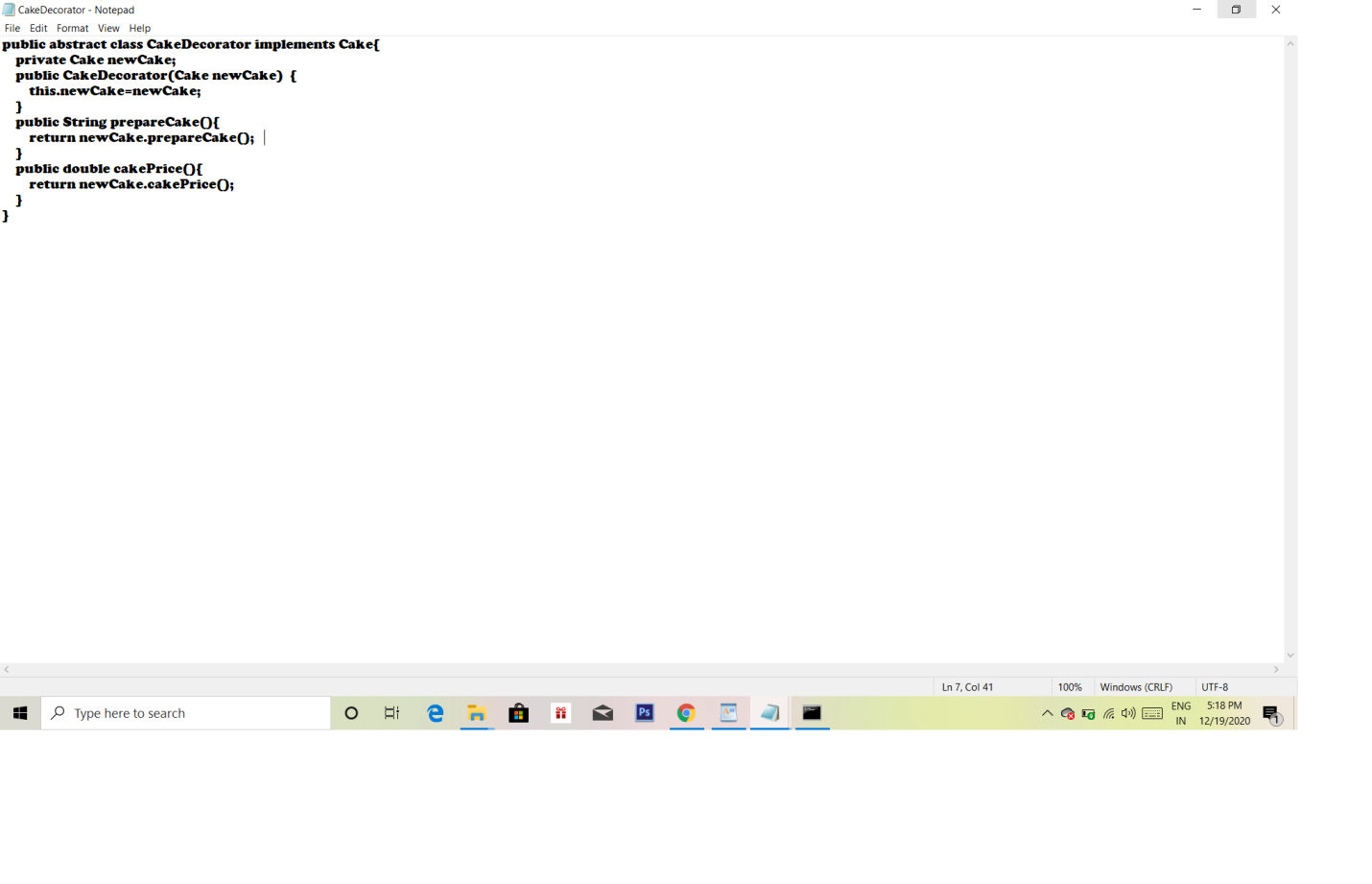
****

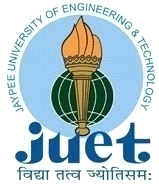
****

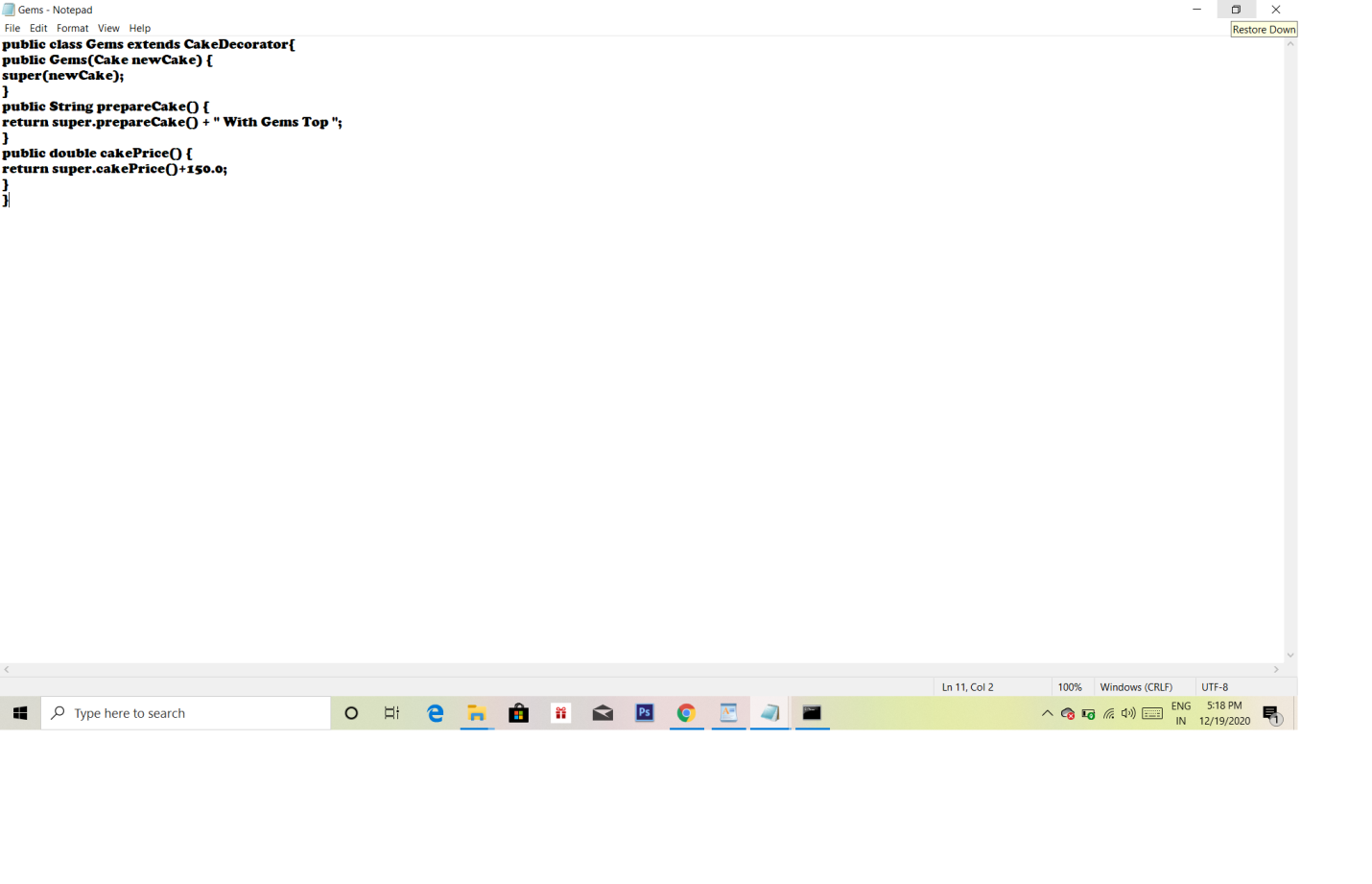
****

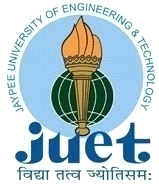
****

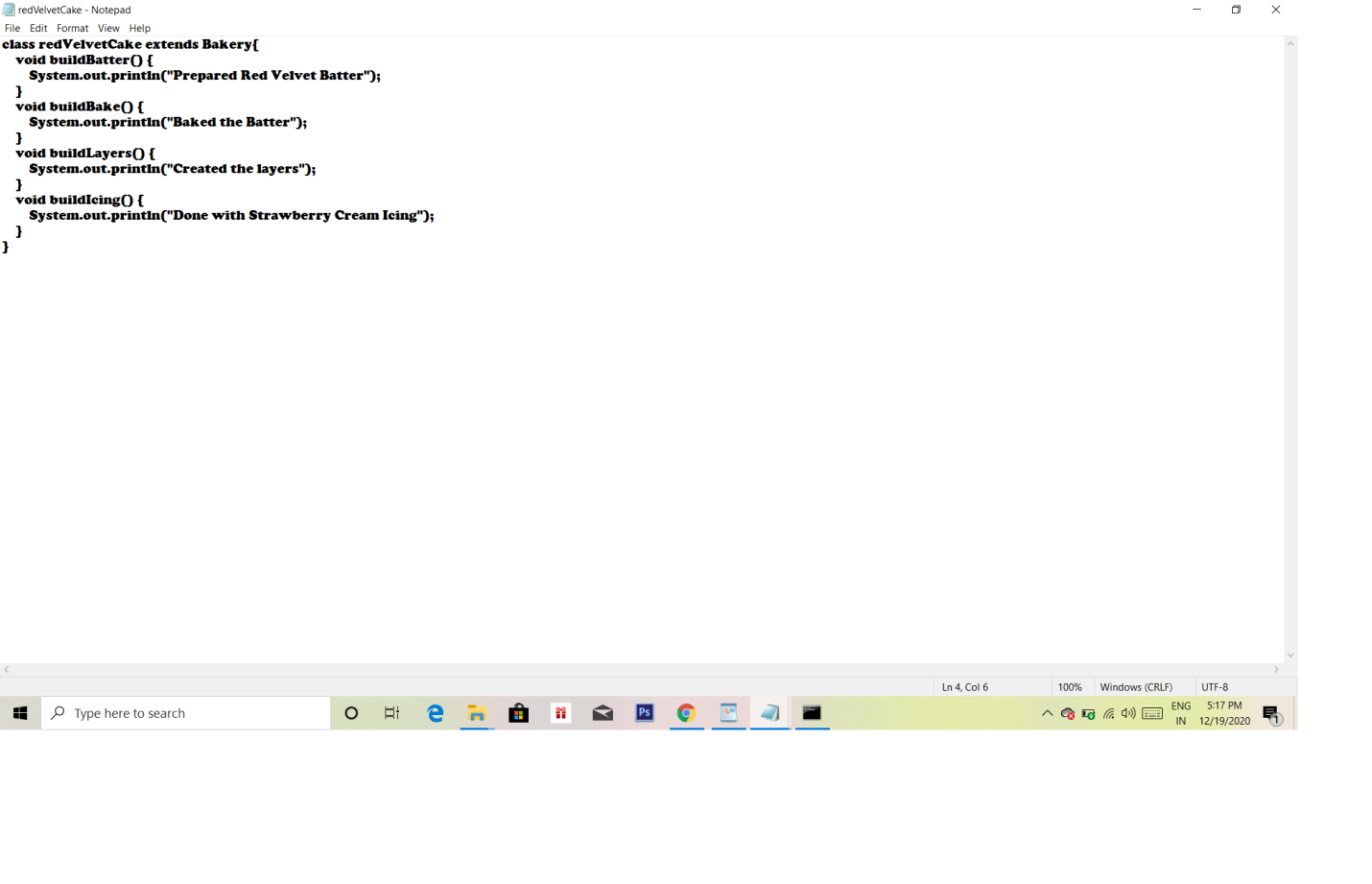
****

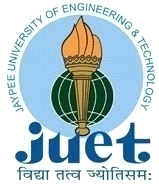
****

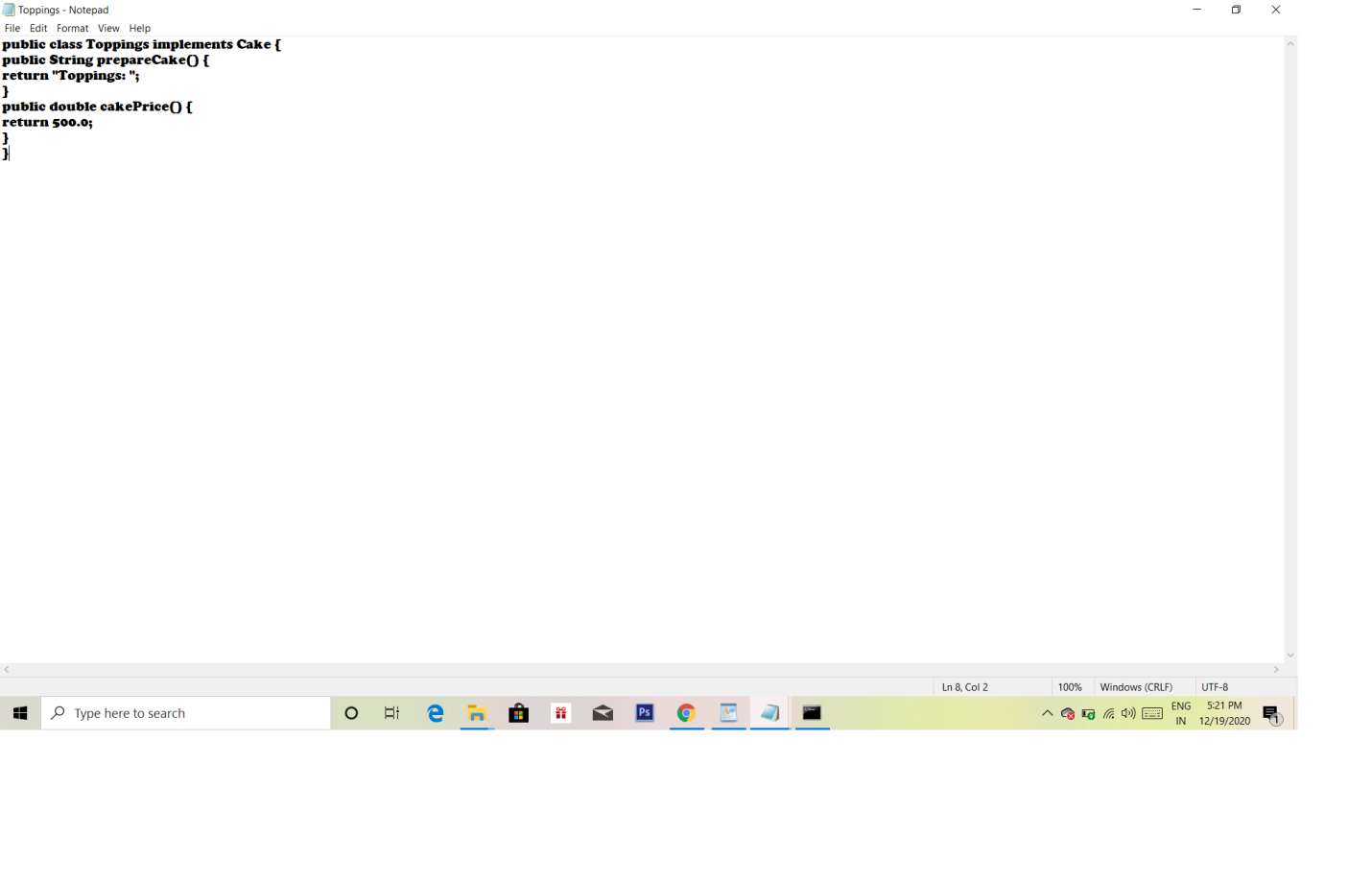
****

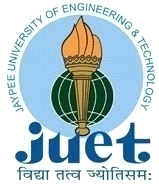
****

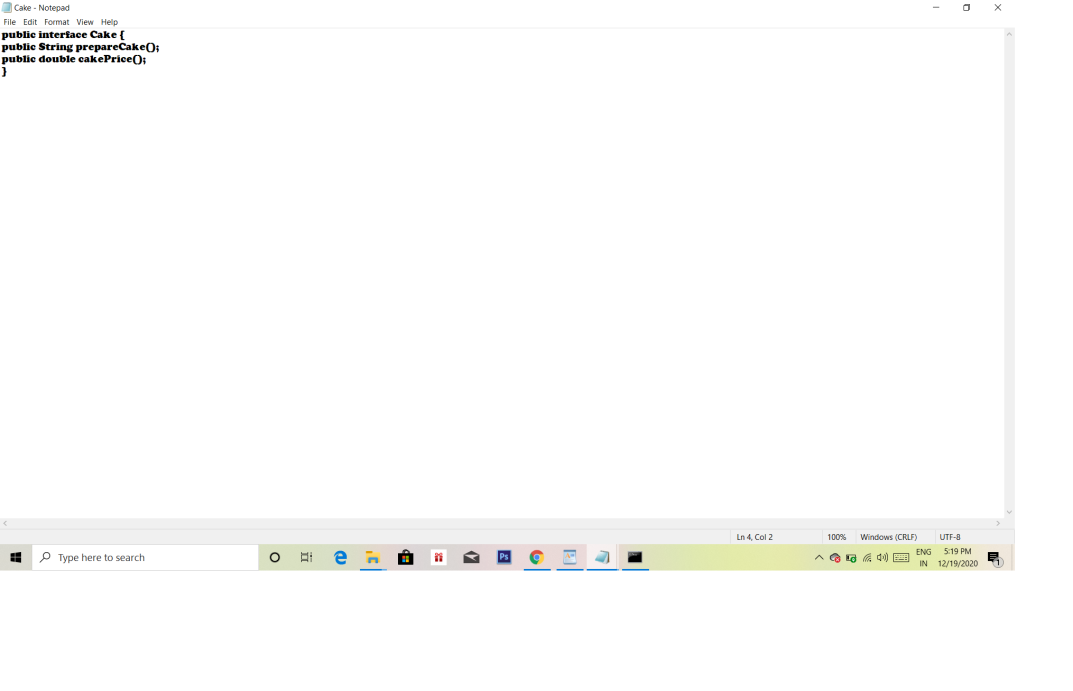
****

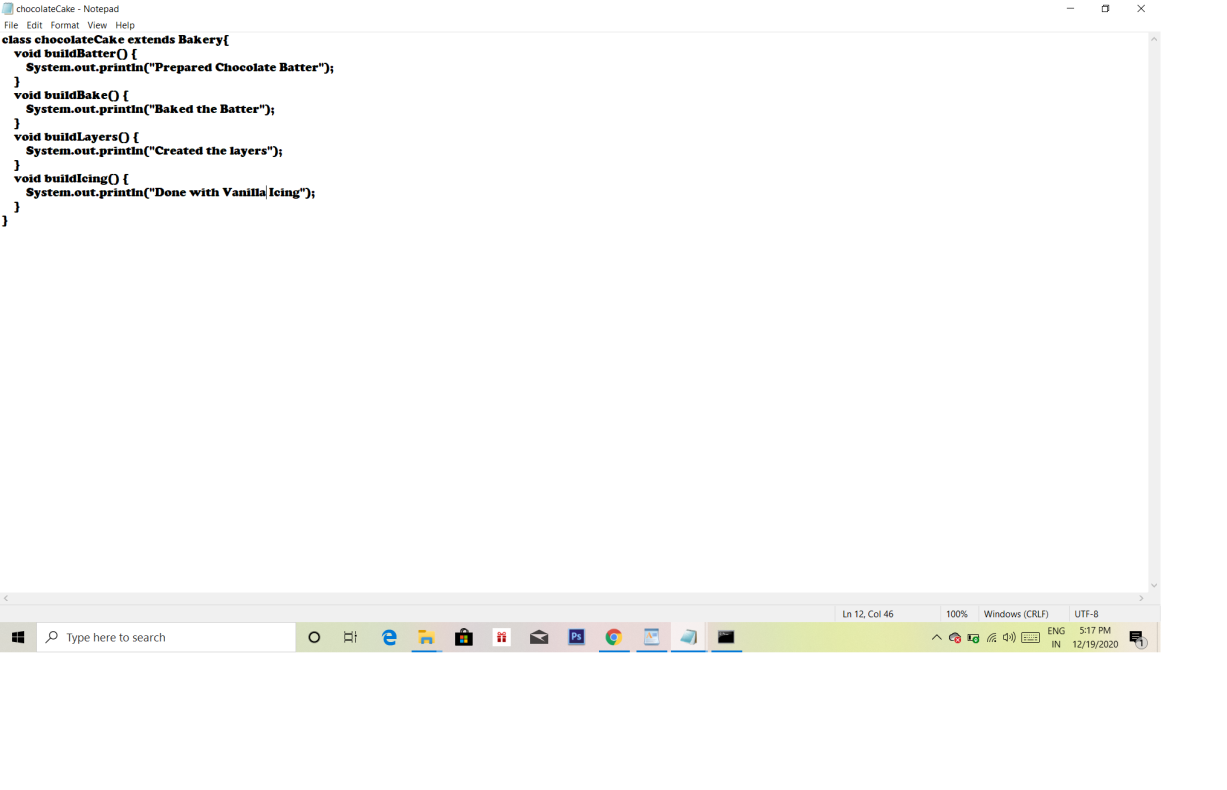
****

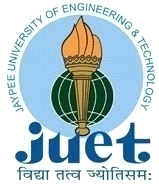
****

****

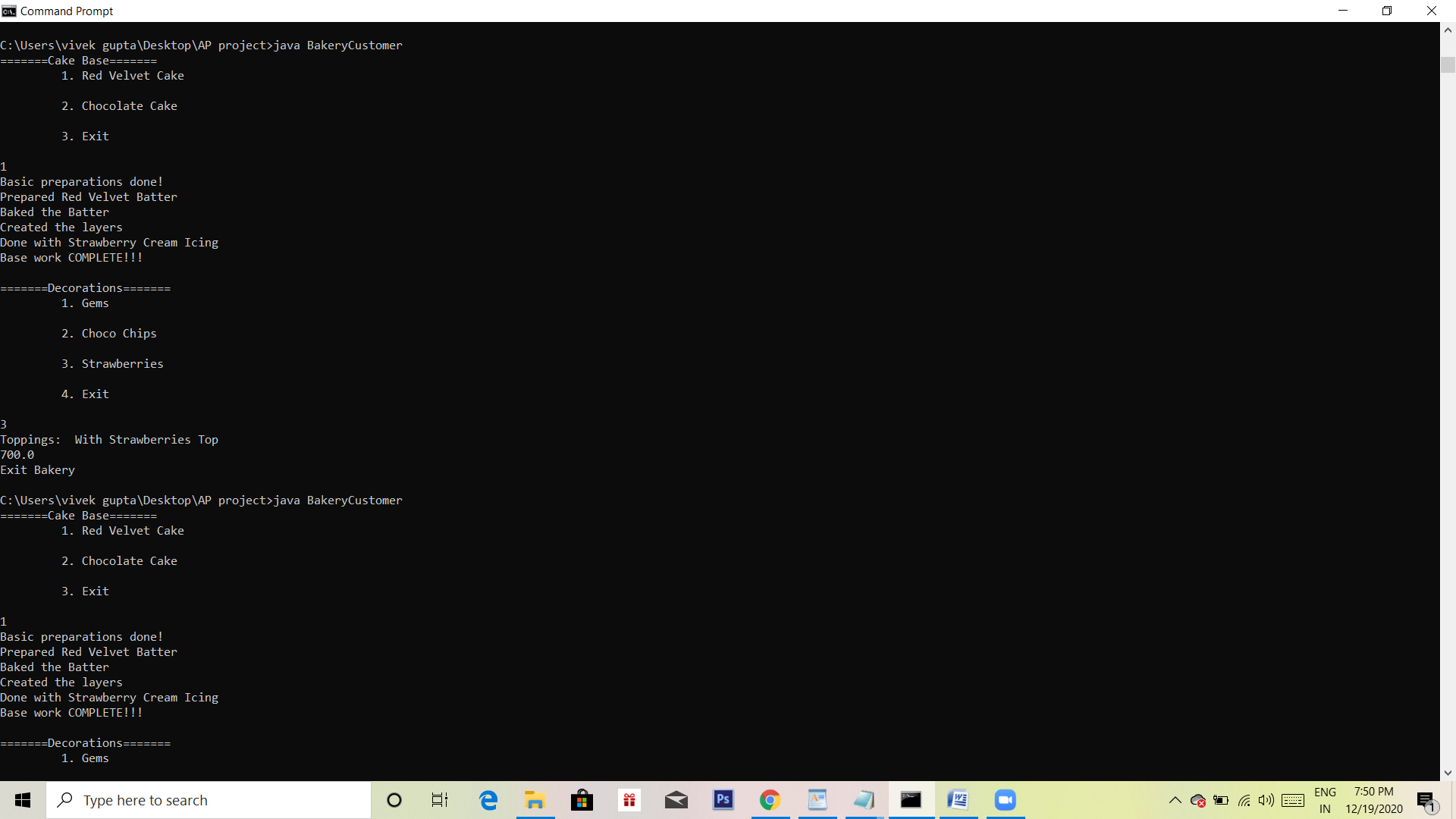
****

****

****

****

**OUTPUT**

****