

-Course: CSC340.03

-Student: Chengkai Yang, SFSU ID: 921572896

-Assignment Number: 02

-Assignment Due Date & Time: 02/17/2022 at 11:59 PM

PART A – OOP Class Design Guidelines

1. Cohesion

[1] A class is a collection of entities with some common characteristics. It is an abstract data type that abstracts entities with the same features. When we discuss cohesion in the java class, it concentrates on two words, “a single entity” and “one class.” In object-oriented programming languages, a class is an abstraction of the properties and behavior of a class of "things.” With high cohesion property, the class will be easier to maintain and change frequently. Suppose we require initializing the method getName in the Person class and output the necessary name in the class showName. Here is the example coding part.

```
class Person{  
    private String name;  
    public String getName(String name){  
        this.name = name;  
        Return name;  
    }  
    Class showName(){  
        Public static void main(String[] args)  
        Name n = new Name();
```

```
        System.out.println(n.getName("George"));
    }
```

In this example, we can find out if we require to change one person's name to output, the only class that can be edited is the showName class. The class Person doesn't need to be edited and increases the cohesion of the program to reach its purpose.

[2] Using the above example, if we require to get one person's name, age, hair color, etc., the best way is to add more classes like Person and display all of the output to one specified class. It can split the several purposes into several small classes to implement. The advantage is evident in making the coding progress clear with high cohesion.

2. Encapsulation

[1] Suppose we see private members in a class. In that case, we should use the keyword “private” to modify them to prevent users from directly accessing and changing the member variables through a class instance member. The private modifier seems to be a password-saving class, which saves the user’s name and password. The class only needs to provide an interface to determine whether the password entered by the user matches the user name. The saved username and password data are all for the user, one person. It is invisible; otherwise, these private data will become transparent and insecure. On the other hand, the private data or methods to modify are usually only used within the class and do not affect the user. If they are exposed to the user, it will make the entire class appear very concise, and the readability of the class will decrease, and it is difficult for users to find the one they need to use among the many interfaces.

[2] We usually provide getter and setter methods for some member variables in Java. As the name mentioned, the former is used to obtain the value of the member variable, and the latter is used to set the value of the member variable. The benefits of using getter and setter come in several ways. On the one hand, they prevent unnecessary information from being exposed to the outside to avoid unnecessary security problems. From another hand, you can also add some legality checks to these getter and setter methods, such as judging to set when setter whether the value meets the requirements. Here is one example to illustrate.

Code Part:

```
Public class personAge{  
    Private int age;  
    Public int getterAge(){
```

```
        Return age;
    }
    Public void setterAge( int t_age){
        If (t_age<=0) return;
        Else age = t_age;
    }
}
```

The coding provides a setter method for the age member variable because age is not negative.

You can judge whether the passed parameter is greater than 0 in the setter function. If the interface is not provided, The member variable age is directly exposed to the user with public modification, and the user may modify age to an illegal value.

3. Instance vs. Static

[1] Because static member variables are shared globally, the same instance is used for all objects of this class, so if you want an object to be shared globally, you should declare it as static. In addition, all Non-static member variables are dependent on the existence of specific objects, and each created object has a corresponding instance, and they do not affect each other.

[2] Typically, for static member variables or methods, because they do not depend on the specific object of the class, they can be called directly through the class name, static variable name, or function name. The advantage of this is that you can see intuitively. The output is a static call to avoid errors in subsequent use.

[3] The constructor may not be used when calling the static member variable because the static member variable does not depend on the specific object. The result obtained may not be expected if the static member variable is initialized during the constructor. Unanimous. The best way is to initialize the static member variable when it is declared to ensure that it is the correct value the first time it is called. In addition, it is best to use setter functions to modify static member variables. As mentioned in Encapsulation, some legality checks can be performed to ensure the correctness of the assignment.

[4] Static and instance programming are separate in some respects, just as we don't instantiate a static class, such as Math, which is a tool class in itself. The meaning of existence is to provide some mathematical methods without storing data. , So there is no need for instantiation.

[5] Non-static member variables cannot be called in static member functions. As mentioned in point [3], on the one hand, the object may not have been initialized when the static function is called. When the object is not initialized directly, it will cause an error. ; On the other hand, because all objects share the static member function, it is impossible to determine which instance

object the member variable it calls belongs to. But for non-static functions, it is natural to call static functions or variables because the meaning of static member variables is that they can be directly accessed.

4. Consistency

[1] For Java programming, camel case naming is usually used; mixed upper and lower case letters form the names of variables and functions. Using this naming method allows programmers to clearly express their naming of classes, functions, and variables. The meaning also enables readers to query the class they want according to this general naming method even if they do not understand it according to the naming convention. When creating a class, the class's member variables are usually placed at the beginning of the class, then its constructor part, and finally other functions. Although Java does not have a strict declaration order in Java when using a certain Before a variable or function, there must be a declaration of the variable or function. Putting the variable at the top of the class has two advantages. One is to make the entire class very concise and avoid interleaving member variables and functions; the other is When querying a variable, you can find it directly at the beginning of the class to avoid the trouble of query and modification.

[2] Keep naming consistency and use similar naming methods for similar operations. You can intuitively guess similar functions or logic based on similar function names. For example, for a function to obtain a member variable, it is usually get+member variable name, and a function for setting a member variable is usually set+member variable name. For example, The function that triggers an event is usually on+event name. This can save time for naming and use the same naming method for similar operations. On the other hand, it also improves the readability of the code.

[3] The programmer usually needs to provide a parameterless constructor when creating a class. In class inheritance, the subclass's constructor needs to call the parent class's constructor. If the user does not explicitly call the parent class's constructor, the compiler will add a parameterless

constructor that calls the parent class by default. However, sometimes the compiler generates a parameterless constructor for the class; if other constructors have been declared in the class, the compiler will not perform this action because the parent class does not have a default parameterless constructor. Therefore, if the no-argument constructor is not supported when creating a class, it needs to be marked explicitly with the document, and then pay attention to the call of the constructor when inheriting the class to prevent incorrect use.

[4] If the programmer does not want a specific class created by the user, you need to declare the constructor as private. There are usually two cases for doing this. One is that, like the Math class, there is no need to have a corresponding entity. Only some static methods need to be provided. The user can use it; the other is a singleton because it is necessary to ensure only one instance of the class in the entire program. It is necessary to declare the constructor as private and get the only instance object in the getInstance method.

5. Inheritance vs. Aggregation

To get through the difference between aggregation, we should figure out the basic concept of inheritance and aggregation. Inheritance refers to the ability of a class (called subclass, subinterface) to inherit the functions of another class (called parent class, parent interface) and add its new functions. In other words, it can be called is-a a relationship. The most common relationship to an interface is identified in Java by the keyword `extends` and is generally not controversial at design time. Moreover, subclasses can automatically inherit the interface of the superclass. When creating an object of a subclass, there is no need to create an object of the parent class.

Aggregation is a particular case of association relationships. It embodies the relationship between the whole and the part and possession, the has-a relationship. At this time, the whole and the part are separable, they can have their cycles, and the parts can belong to multiple overall objects and can also be shared by multiple overall objects; for example, the relationship between computer and CPU, company and employee, etc. I will use two examples to illustrate the difference.

Code Part:

Inheritance:

```
class A {  
    String code;  
    public void setCode(String code) {  
        this.code = code;  
    }  
}
```

```

public String getCode() {
    return this.code;
}
}

```

```

class B extends A {
    public void print() {
        System.out.println(code);
    }
}

```

The inheritance is easier to understand class extends from A. It will set String code first and then get String code and then output the words in the main class. It is apparent is-a relationship.

Aggregation:

```

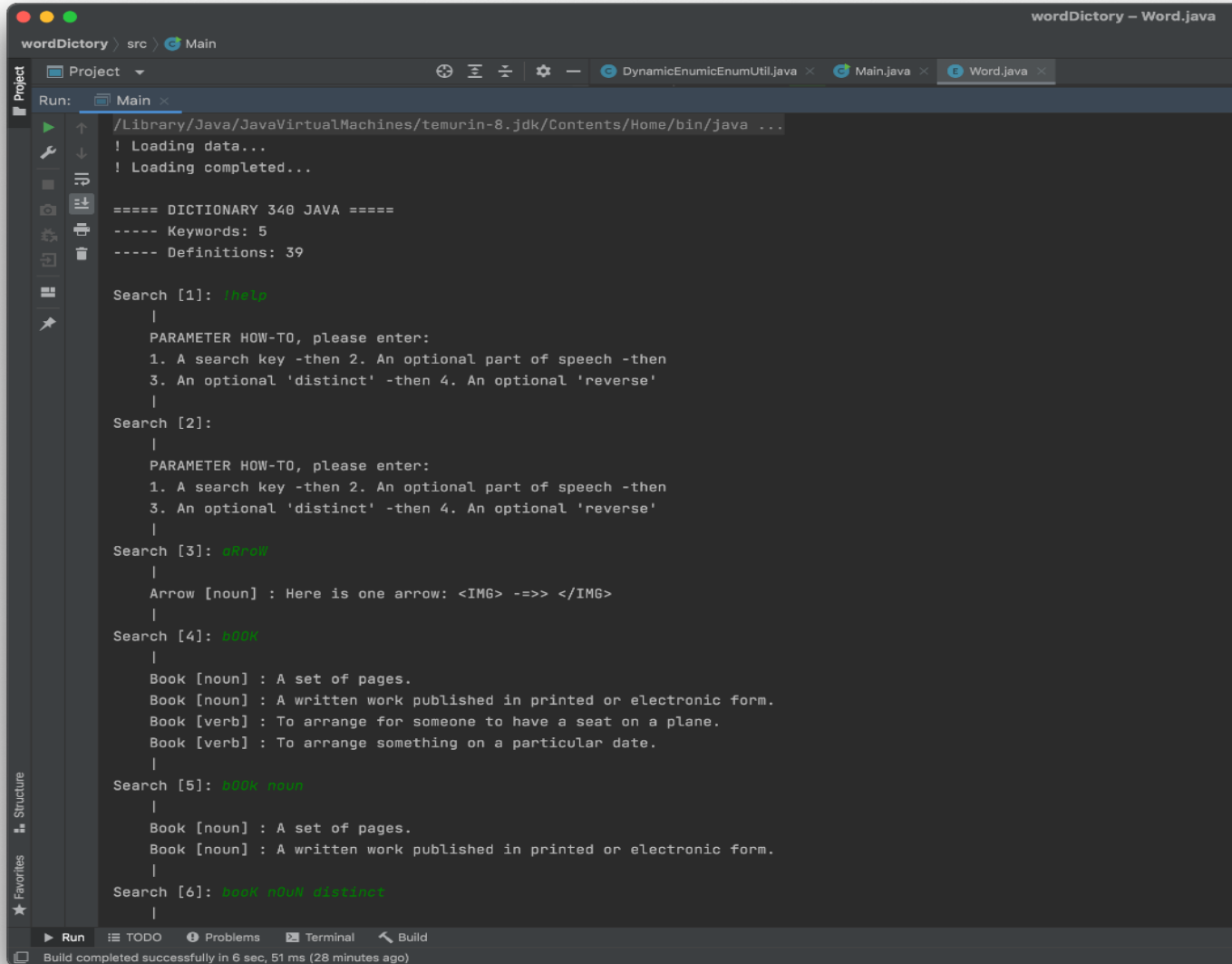
Class family{
    String name;
    Address address;
    Public family( String name, Address address){
        This.name = name;
        This.address = address;
    }
    Public static void main(String[] arg){
        //Output name and address
    }
}

```

Suppose we initialize the family name and address in the class family; we discover that in class inheritance the relationship between family name and address is not an inheritance relationship. It is time to use aggregation because relationships cannot maintain through whole cycles.

PART B – Java Programming, Data Structures, and Data Design

Code output:

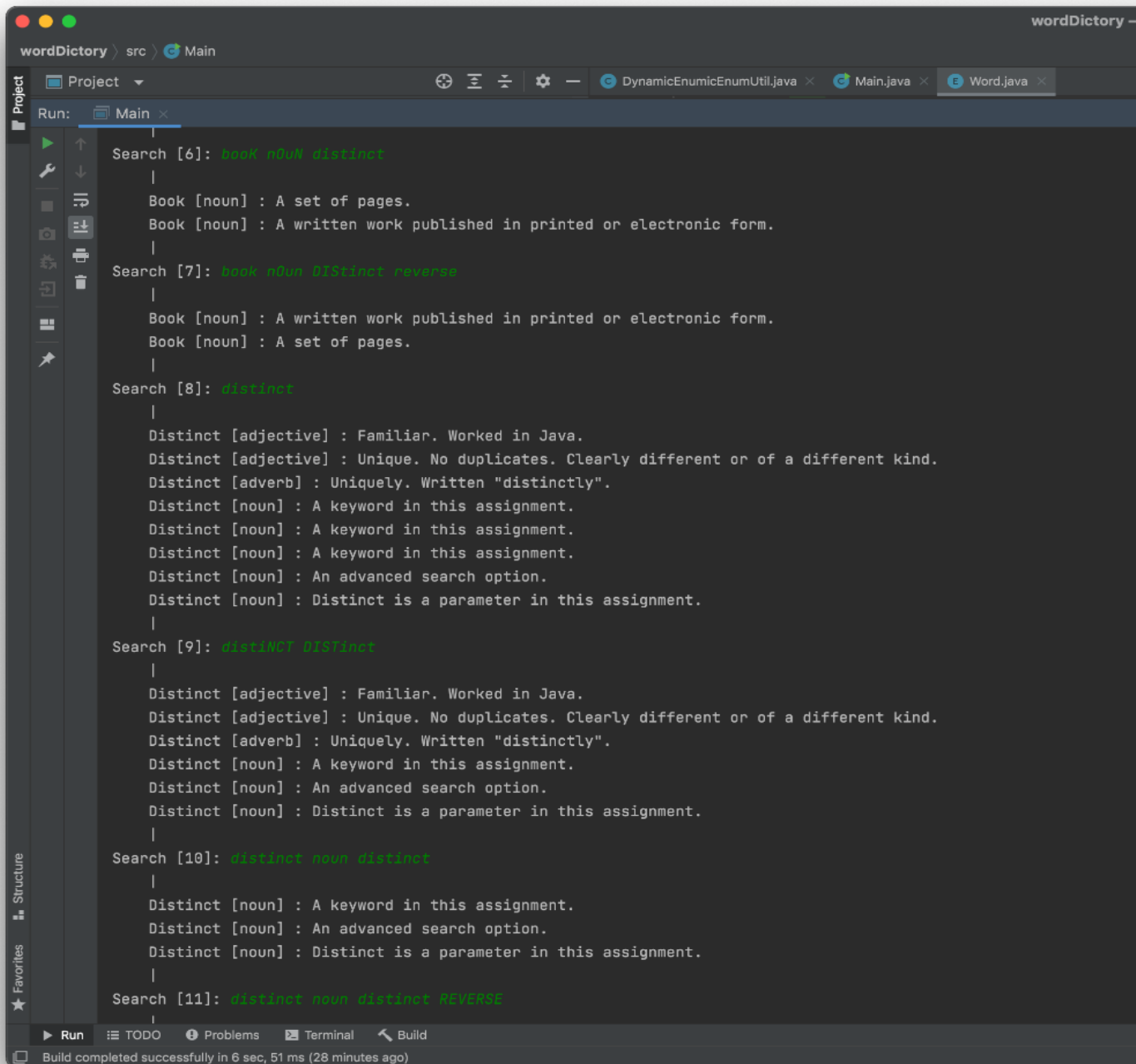


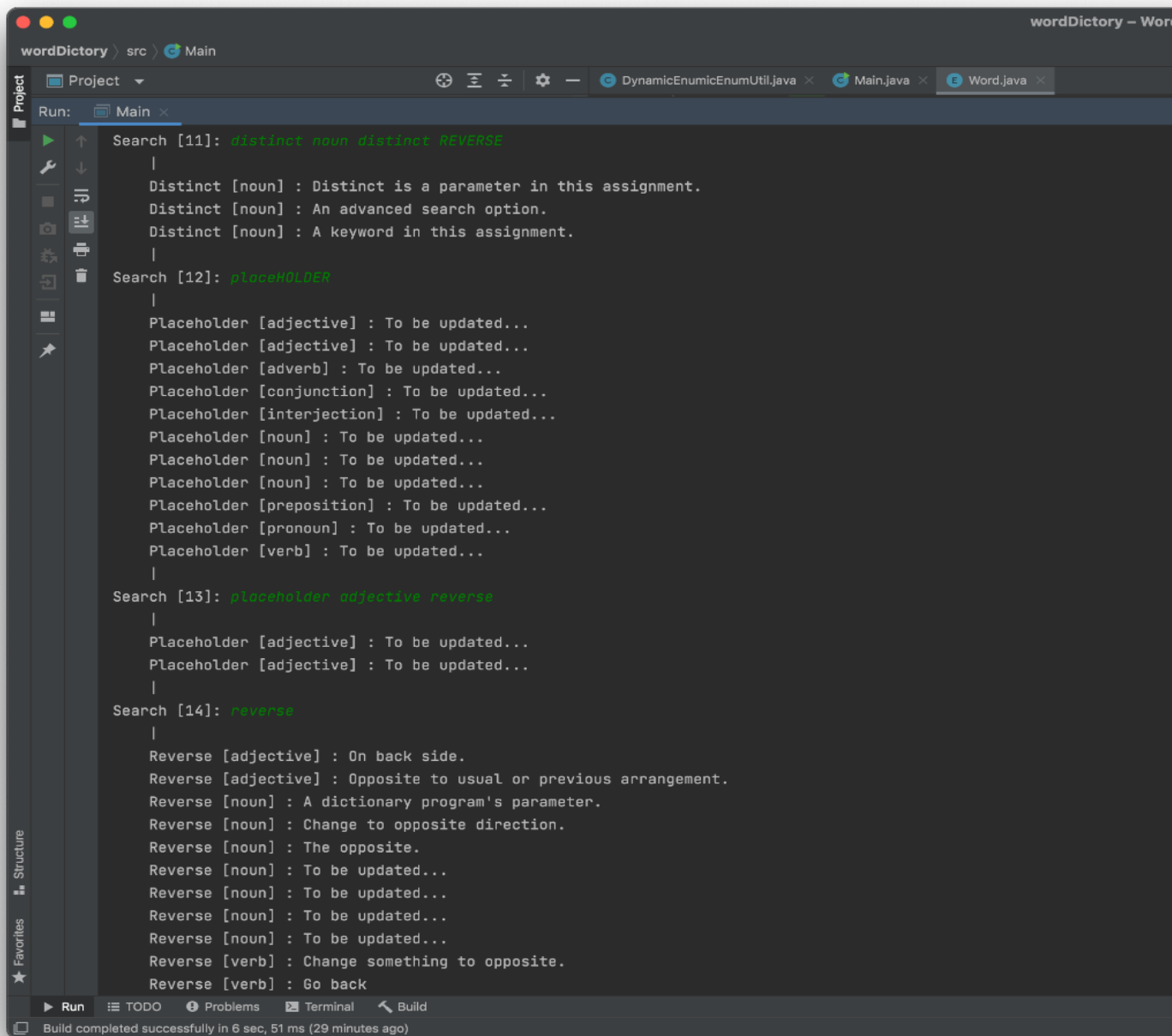
```
wordDictionary - Word.java
wordDictionary > src > Main
Run: Main x
/Library/Java/JavaVirtualMachines/temurin-8.jdk/Contents/Home/bin/java ...
! Loading data...
! Loading completed...

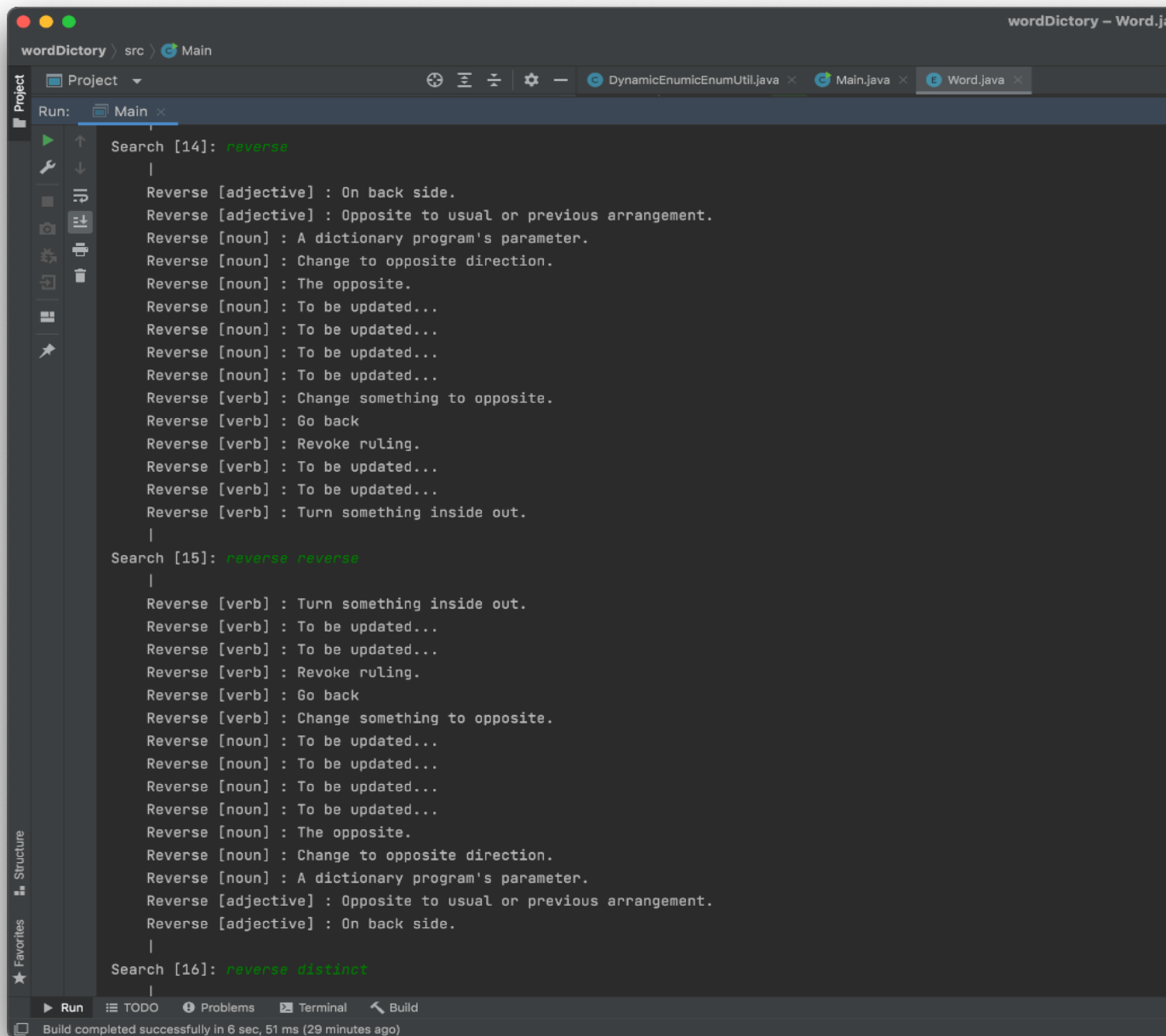
===== DICTIONARY 340 JAVA =====
----- Keywords: 5
----- Definitions: 39

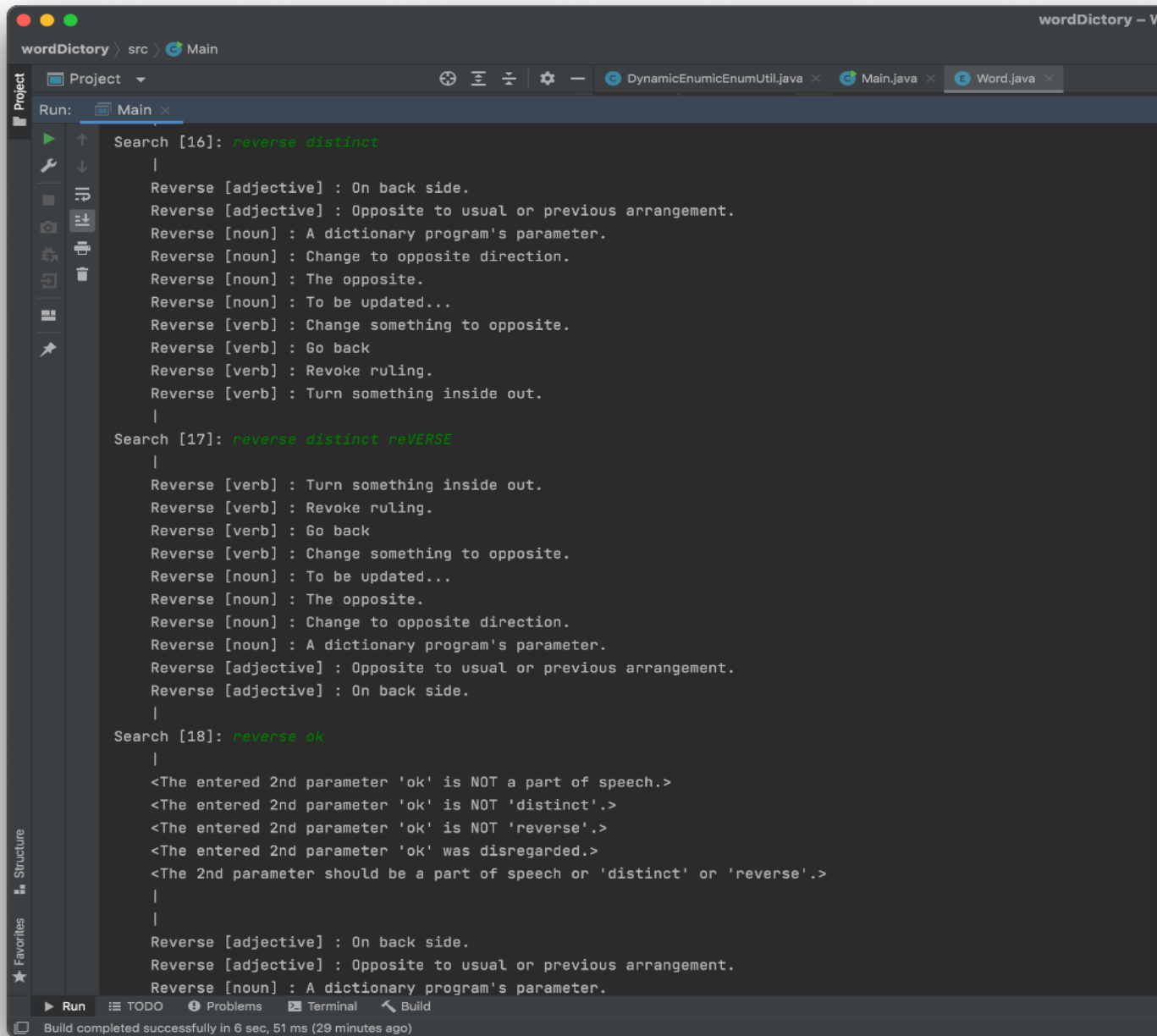
Search [1]: help
|
PARAMETER HOW-TO, please enter:
1. A search key -then 2. An optional part of speech -then
3. An optional 'distinct' -then 4. An optional 'reverse'
|
Search [2]:
|
PARAMETER HOW-TO, please enter:
1. A search key -then 2. An optional part of speech -then
3. An optional 'distinct' -then 4. An optional 'reverse'
|
Search [3]: arrow
|
Arrow [noun] : Here is one arrow: <IMG> -==> </IMG>
|
Search [4]: book
|
Book [noun] : A set of pages.
Book [noun] : A written work published in printed or electronic form.
Book [verb] : To arrange for someone to have a seat on a plane.
Book [verb] : To arrange something on a particular date.
|
Search [5]: book noun
|
Book [noun] : A set of pages.
Book [noun] : A written work published in printed or electronic form.
|
Search [6]: book noun distinct
|
```

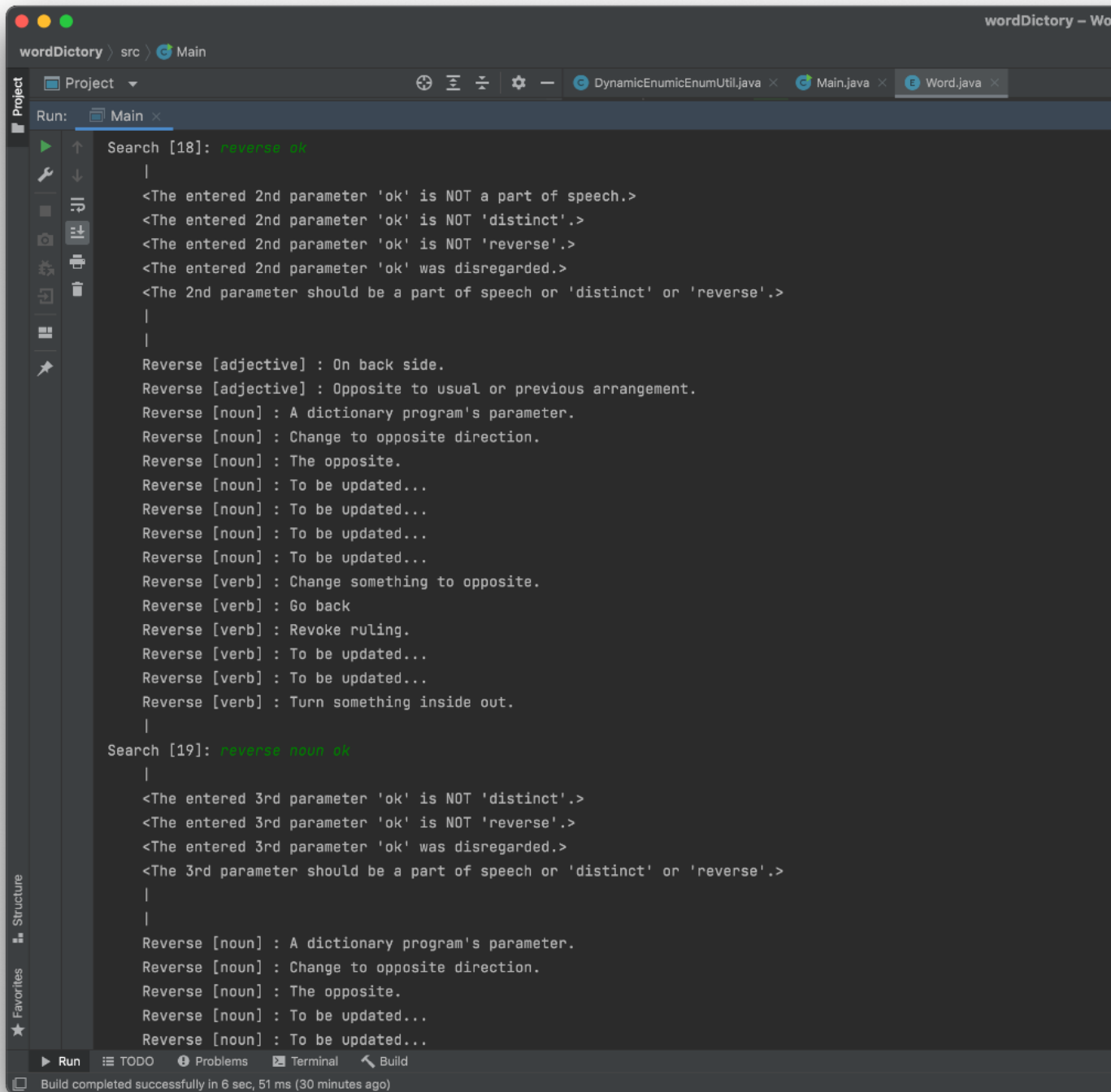
Build completed successfully in 6 sec, 51 ms (28 minutes ago)

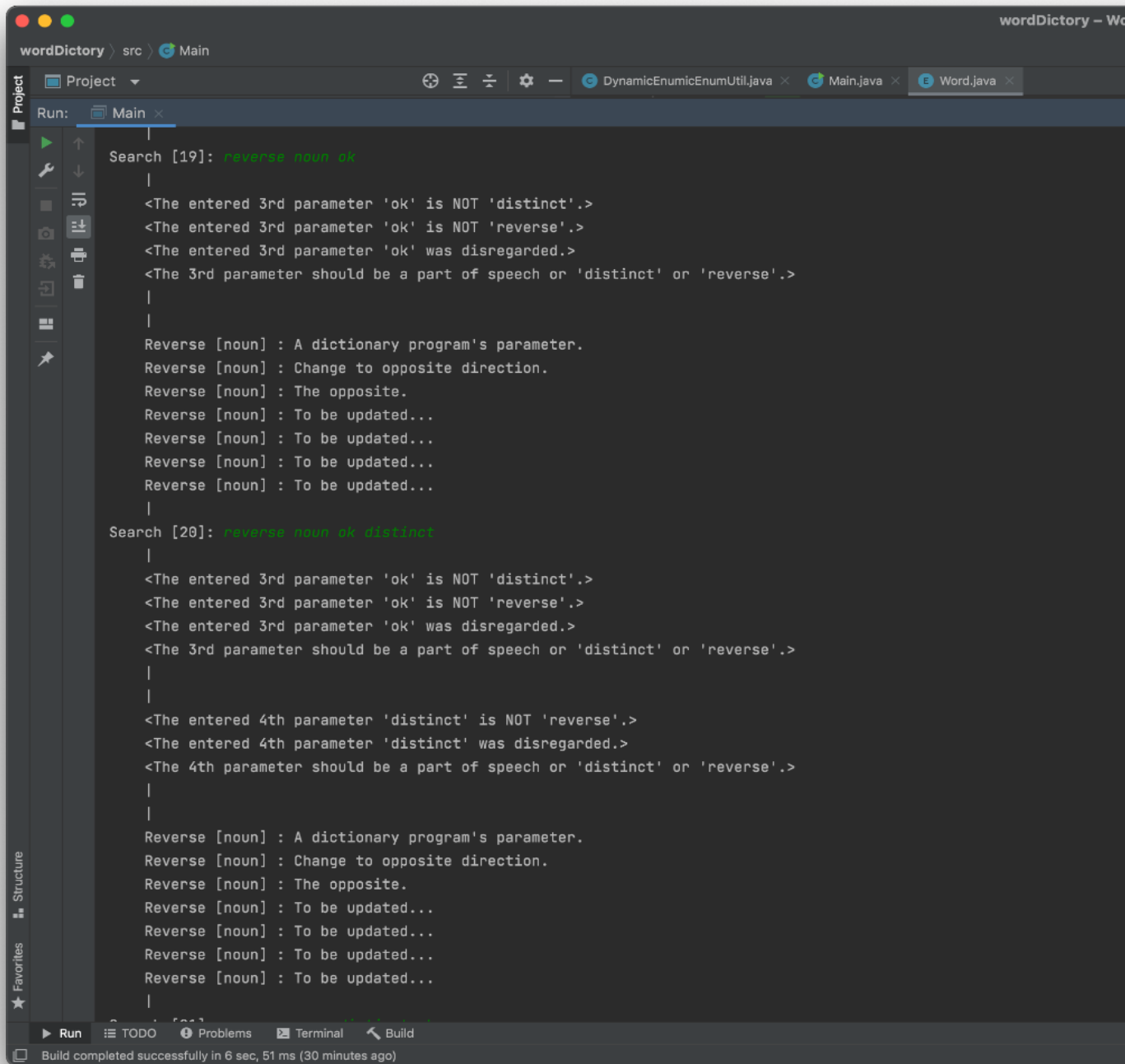


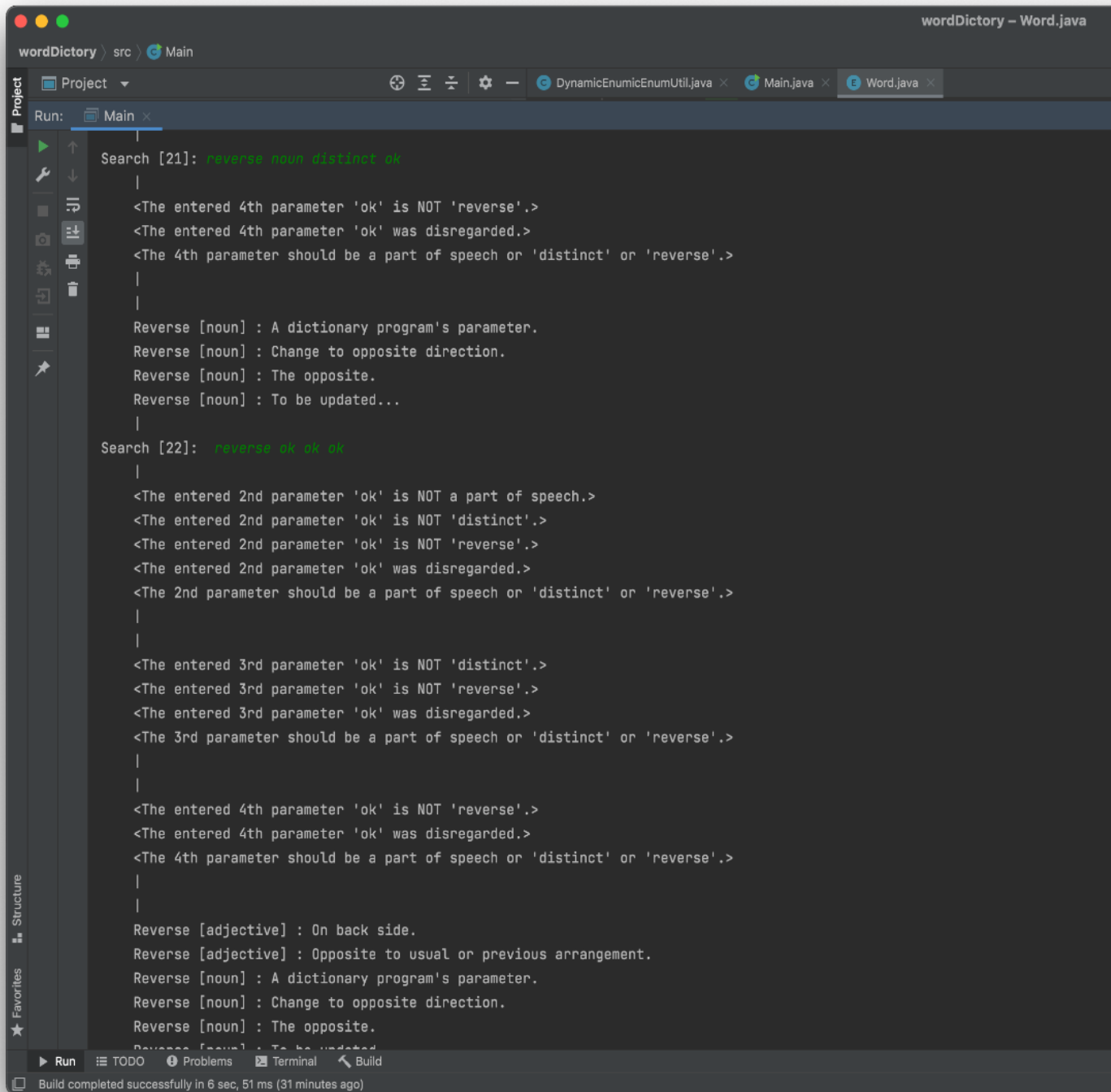


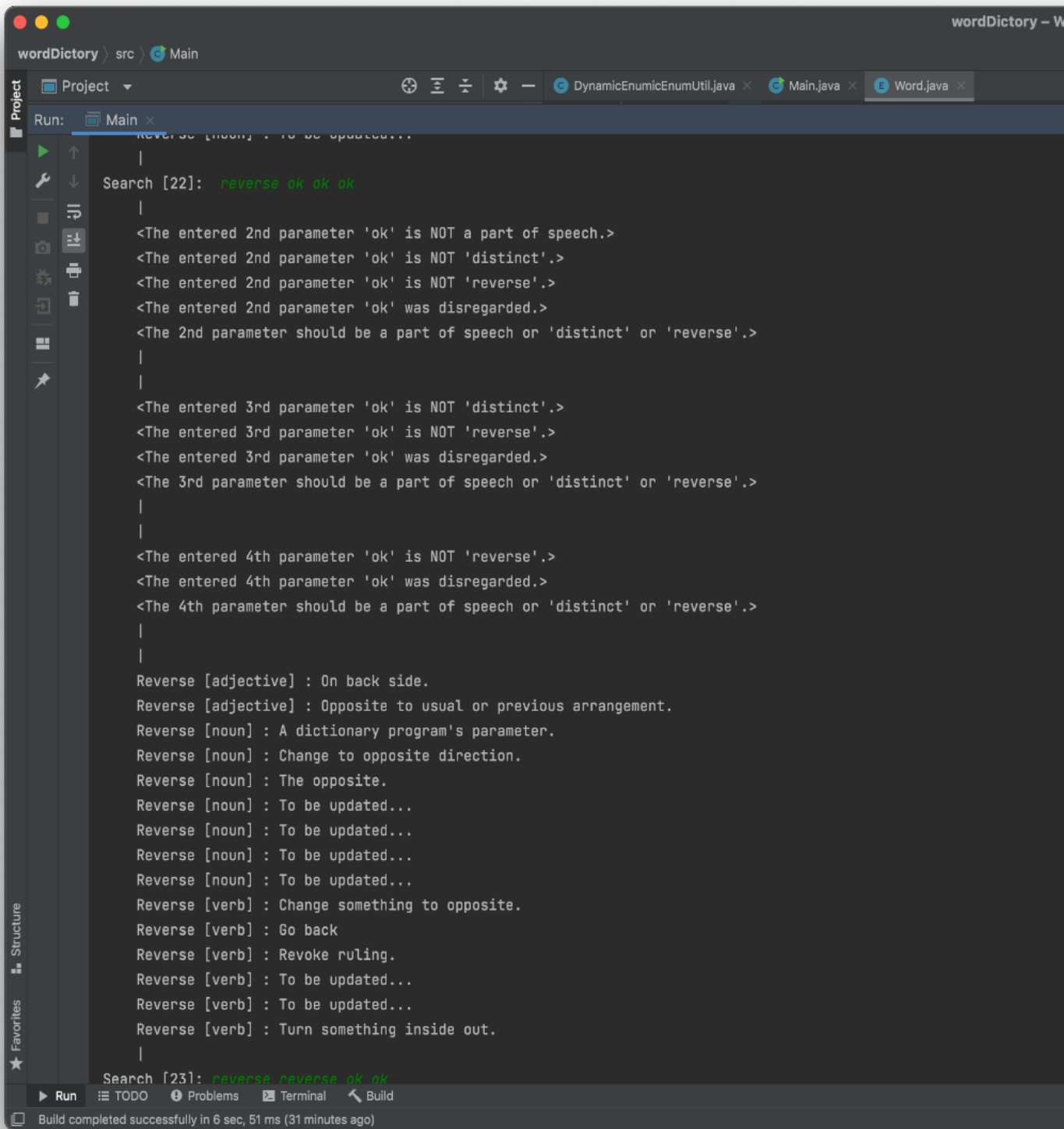












|
Search [23]: reverse reverse ok ok

|
<The entered 3rd parameter 'ok' is NOT 'distinct'.>
<The entered 3rd parameter 'ok' is NOT 'reverse'.>
<The entered 3rd parameter 'ok' was disregarded.>
<The 3rd parameter should be a part of speech or 'distinct' or 'reverse'.>

|
|
<The entered 4th parameter 'ok' is NOT 'reverse'.>
<The entered 4th parameter 'ok' was disregarded.>
<The 4th parameter should be a part of speech or 'distinct' or 'reverse'.>

|
|
Reverse [verb] : Turn something inside out.
Reverse [verb] : To be updated...
Reverse [verb] : To be updated...
Reverse [verb] : Revoke ruling.
Reverse [verb] : Go back
Reverse [verb] : Change something to opposite.
Reverse [noun] : To be updated...
Reverse [noun] : To be updated...
Reverse [noun] : To be updated...
Reverse [noun] : To be updated...
Reverse [noun] : The opposite.
Reverse [noun] : Change to opposite direction.
Reverse [noun] : A dictionary program's parameter.
Reverse [adjective] : Opposite to usual or previous arrangement.
Reverse [adjective] : On back side.

Search [24]: *reverse distinct reverse ok*

|

<The entered 4th parameter 'ok' is NOT 'reverse'.>

<The entered 4th parameter 'ok' was disregarded.>

<The 4th parameter should be a part of speech or 'distinct' or 'reverse'.>

|

|

Reverse [verb] : Turn something inside out.

Reverse [verb] : Revoke ruling.

Reverse [verb] : Go back

Reverse [verb] : Change something to opposite.

Reverse [noun] : To be updated...

Reverse [noun] : The opposite.

Reverse [noun] : Change to opposite direction.

Reverse [noun] : A dictionary program's parameter.

Reverse [adjective] : Opposite to usual or previous arrangement.

Reverse [adjective] : On back side.

|

Search [25]: *reverse distinct reverse ok ok*

|

PARAMETER HOW-TO, please enter:

1. A search key -then 2. An optional part of speech -then

3. An optional 'distinct' -then 4. An optional 'reverse'

|

Search [26]: *reverse adverb*

|

<NOT FOUND> To be considered for the next release. Thank you.

|

|

PARAMETER HOW-TO, please enter:

1. A search key -then 2. An optional part of speech -then

3. An optional 'distinct' -then 4. An optional 'reverse'

Program Analysis to Program Design

In this design, we are using enumerations to store data program Implementation. However, the method of using enumeration has great limitations, it's hard to enumerate everything, but we have to do that in the course. So when I am designing an enumeration class: Word, I designed three member variables to store different messages: keyword, speech, and definition. As I mentioned earlier, we can't enumerate everything. So to dynamically add enumerations to enumerations, I use reflection. This allows us to add the contents of the data file to the enumeration class, The data content should be loaded every time the program is started, and the data content should be clear after running. But in the actual operation of the dictionary, I used ArrayList many times. Then I store keywords, speeches in different ArrayList, so it's easy to let me know if we have the same keyword or speech or definitions, and it's easy to count the times they occurrences.