# Assignment 03 Application Design: Patterns and Frameworks 44642

Answer **all** the questions below. In your answer for each question explain with sample code or image whichever is preferable.

1. What are generics?

Generics are a feature of a computer language that lets you write code that works with many different types of data without having to know the exact type at compile time. Generics are a way to tell a class or method what kinds of things it can work with.

Generics let you write code that can work with any type of data instead of being tied to a specific type. This makes your code more flexible and useful. For example, a generic sorting algorithm can work with any type of data, such as integers, strings, or custom objects, without having to write different code for each type of data.

In Java, generics are made possible by type arguments, which are given after the name of a class or method inside angle brackets (<>).

Graphical user interface, text

Description automatically generated

1. Can we change the scope of the overridden method in the subclass for private, public, default and protected? Explain how can it be changed for each scope?

In Java, you can't make the access level (scope) of a modified method in a subclass more limited than the access level of the method in the superclass. You can only change the access level, which means you can make a method that was private or public in the superclass public in the subclass.

Graphical user interface, text, application

Description automatically generated

1. What is the covariant return type?

Covariant return type is a Java 5 feature that lets a subclass method return a type that is more specific than the type returned by the method that it overrides in the superclass.

It means that if a method in a superclass returns a type T, a method in a subclass that modifies it can return a subtype of T.

A screenshot of a computer

Description automatically generated

Graphical user interface, text, application

Description automatically generated

1. Can we override the static and private methods? Why?

Static methods are part of the class, not a copy of the class, so you can't change them in Java. At compile time, static methods are handled based on the type of the reference variable, not based on the type of the object at run time. Because of this, you can't change how a static method works in a clone.

On the other hand, you can't change secret methods because no one outside the class can see them. Private methods don't get passed down to subclasses, so they can't be changed. If a subclass creates a method with the same name as a private method in the superclass, it is just a new method that happens to have the same name as the private method in the superclass.

In short, we can't change basic and private Java methods. The reason is that static methods are resolved at compile time and belong to the class, while secret methods are not visible outside of the class and are not inherited by the subclass.

Graphical user interface, application

Description automatically generated

1. Difference between String Buffer and StringBuilder?

StringBuffer and StringBuilder are classes in Java that describe sequences of characters that can be changed. Both classes have ways to change the contents of the series, but the way they handle synchronization is different.

The main difference between StringBuffer and StringBuilder is that StringBuffer is thread-safe, which means it can be used safely in a multi-threaded system, while StringBuilder is not thread-safe.

Key changes between StringBuffer and StringBuilder are as follows:

Thread-safety: As stated, StringBuffer is thread-safe because its methods are synchronized, which means that only one thread at a time can access the object. StringBuilder, on the other hand, is not thread-safe because its methods are not synced. This makes it faster in environments with only one thread.

Performance: StringBuilder is faster than StringBuffer in a single-threaded context because it is not synchronized. But in a multi-threaded system, the way StringBuffer synchronizes can slow things down.

Constructors: The constructors for both classes are similar, but StringBuilder has one more that takes a starting capacity as an argument. This can be helpful if you know how many characters you will be adding to the series.

A screenshot of a computer

Description automatically generated

1. Difference between String class and String Buffer?

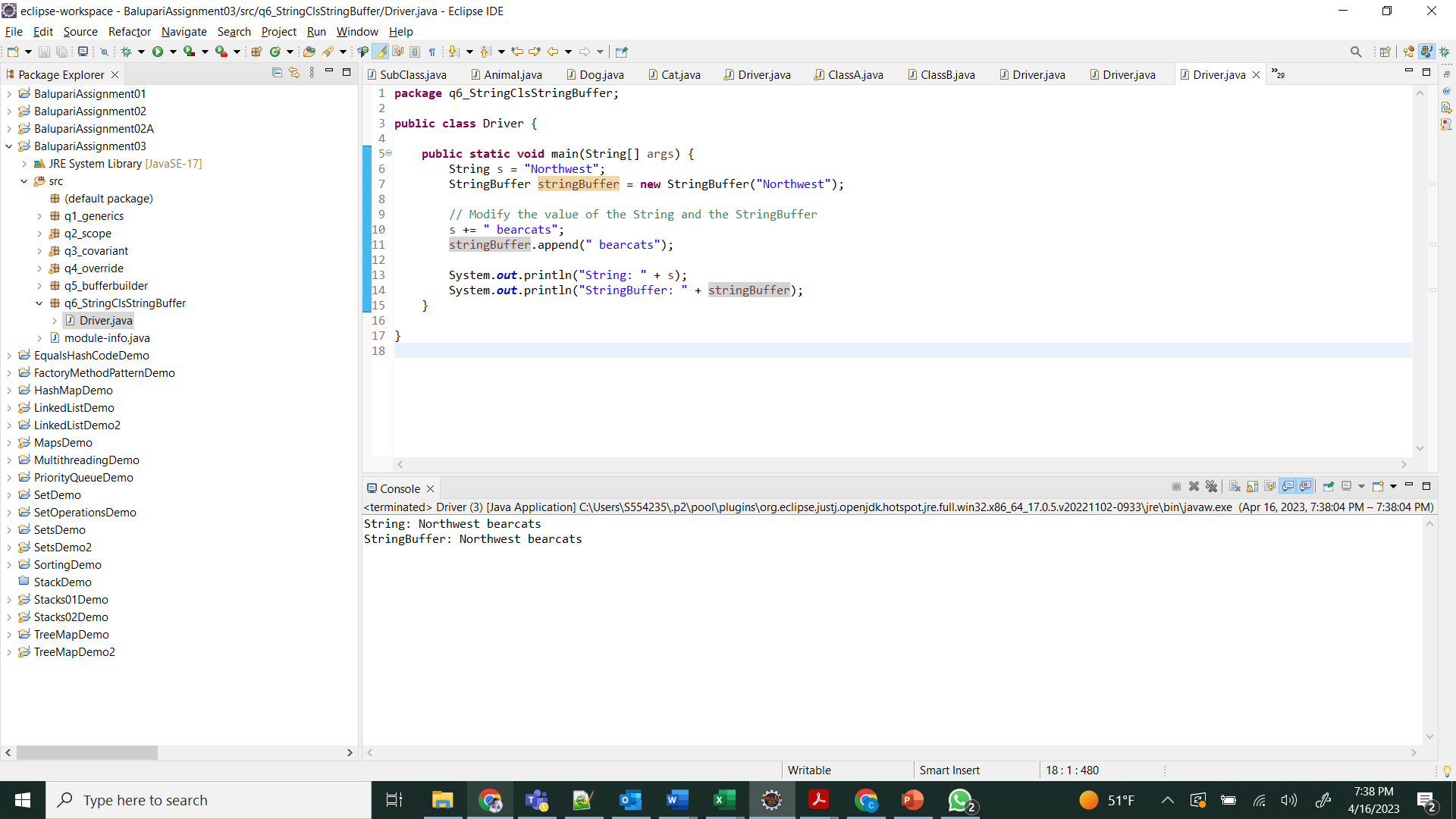
String and StringBuffer are both classes in Java that describe strings of characters, but they are not the same.

Here are some of the main ways in which String and StringBuffer are different:

String is immutable, which means that once it is made, the value of a String object cannot be changed. Every action that looks like it changes a String actually makes a new String object. StringBuffer, on the other hand, can be changed, which means that its value can be changed.

Performance: Because String is fixed, operations that involve reading and comparing strings are fast, but operations that involve changing strings can be slow. For example, if you need to join together a lot of strings, it is faster to use StringBuffer because it can change the value right where it is. On the other hand, String is better to use if you need to do many read-only actions on a string.

Thread-safety: String is thread-safe because it can't change, so it can be used by multiple threads at the same time. On the other hand, StringBuffer is not thread-safe because it is mutable. This means that if more than one thread accesses it at the same time, its value might change in a way that is not intended. Java has a version of StringBuffer that doesn't interfere with other threads. This version is called StringBuilder.



1. Can we declare constructor as final?

No, we cannot create constructor as final because we cannot initial the object while creating the object.

Graphical user interface, text, application, Word

Description automatically generated

1. Can we have try without catch block in java?

Yes, we can have a try block in Java without a catch block, but we must also have a finally block.

The try block is used to wrap up a section of code that could cause an error. In a catch block, an exception can be caught and dealt with if it is made. The finally block is used to wrap up code that should be run whether or not an exception is thrown.

Graphical user interface, text, application

Description automatically generated

1. What is try with the resource?

"Try with resources" is a function of the Java 7 programming language that allows resources that are being used in a try block to be closed automatically.

Before this feature was added, a try-finally block was often used to make sure that resources like files, sockets, and database connections were stopped correctly, even if an exception was thrown. With "try with resources," these resources are automatically closed.

Graphical user interface, text, application

Description automatically generated

1. Can we modify the throws clause of the superclass method while overriding it in the subclass?

No, we can't change the throws part of the method in the superclass while modifying it in the subclass.

In the throws section of a method, you list the errors that method can throw. When a subclass replaces a method of its superclass, the new method must have the same signature as the original method. This means that it must have the same method name, argument types, and return type. This includes the throws clause: the method being overridden can't throw any new checked exceptions that weren't defined in the throws clause of the original method.

But the overriding method can choose not to throw some or all of the exceptions that are declared in the throws clause of the original method, or it can throw a subclass of an exception that is declared in the throws clause of the original method.

A screenshot of a computer

Description automatically generated

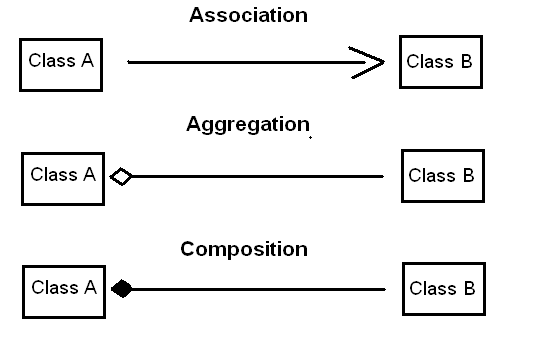
1. What is an association, aggregation, and composition in UML?

In the Unified Modeling Language (UML), there are three different ways that classes or items in a system can be related to each other. These are association, aggregation, and composition.

Association: An association is a link between two or more classes that doesn't depend on the other classes. It is shown by a line between two classes, with or without an arrow to show which way the connection goes. A university may have a relationship with a school, for example.

Aggregation: Aggregation is a special kind of connection in which one class is a part of another class. It shows a "has-a" relationship. The relationship between the whole and the parts is smaller than the relationship between the whole and the composition. Aggregation is shown by a diamond shape on the class that holds the parts and a line connecting the diamond to the class that represents the whole. For example, a car and its wheels have an aggregation link.

Composition: One class is made up of one or more other classes. This is a greater form of aggregation. It shows a "contains-a" connection, where the parts can't exist without the whole. Composition is shown by a filled diamond shape on the class that holds the parts, with a line connecting the diamond to the class that shows the whole. For example, a car and its engine have a composition link.



1. Difference between final, finally and finalize()?

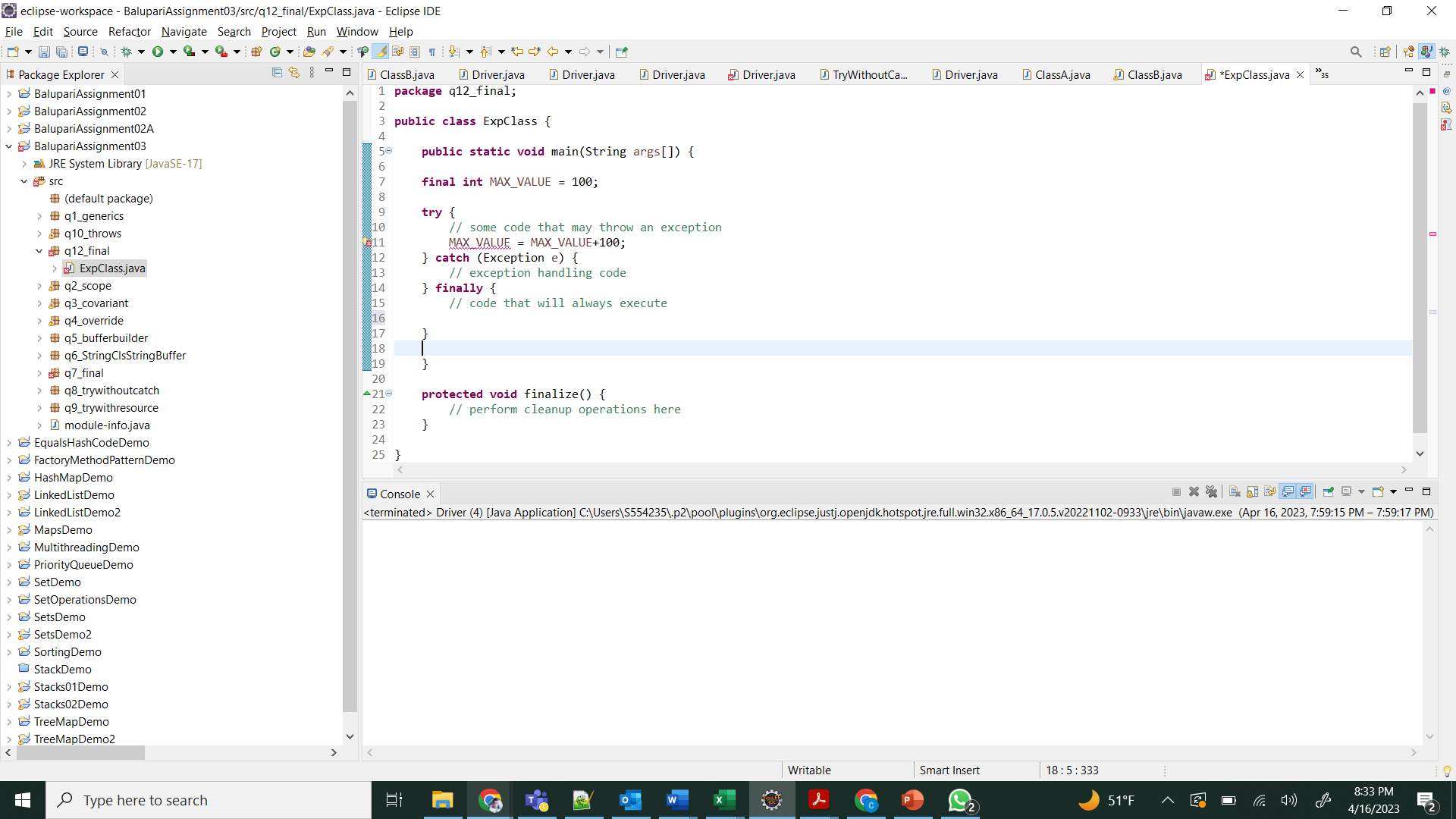
In Java, final, finally, and finalize() are all different.

final is a Java term that is used to stop a variable, method, or class from being changed. If a variable is marked as "final," its value can't be changed after it has been set. If a method is marked as "final," subclasses cannot change it. If a class is marked as "final," other classes can't add to it.

finally: In Java, finally is a block that makes sure a piece of code is always run, even if an exception is thrown. The finally block goes after the try and catch blocks. It has code that must be run no matter what happens in the try and catch blocks.

finalize() is a Java method that is called when an object is about to be destroyed by the trash collector. It can be used to clean up the object before it is killed. For example, it can be used to release resources or close files. The Object class defines the finalize() method, so any class can change it. But you shouldn't count on the finalize() method to clean up because you can't be sure it will be called at a certain time or even at all.

In short, final is used to prevent changes from being made, finally is used to make sure code is always run, and finalize() is a method that the trash collector uses to clean up.



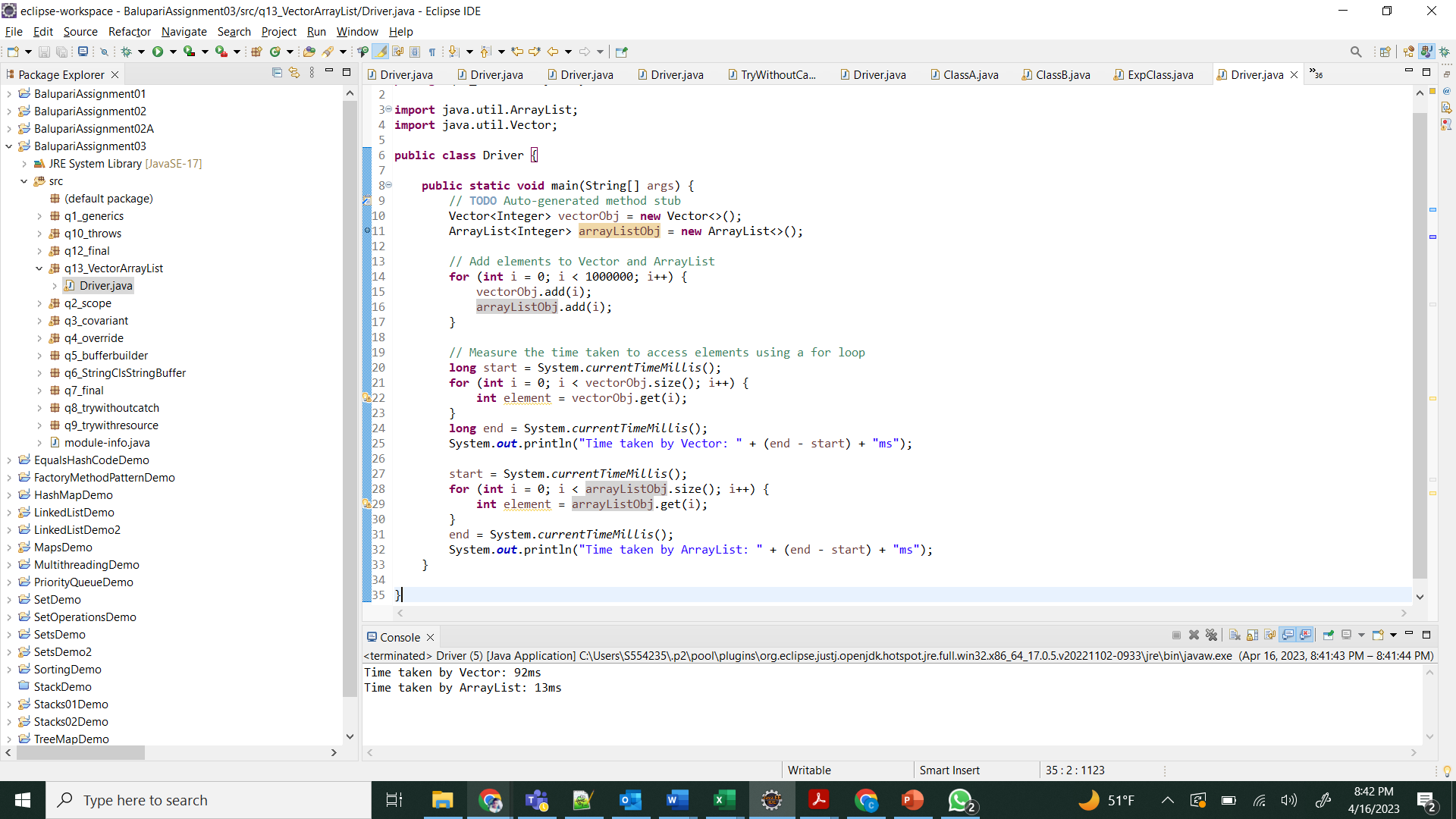
1. Difference between Vector and ArrayList?

Vector and ArrayList are both Java classes that implement the List interface, but they are not the same.

Vector is synchronized, which means that more than one thread can't handle the same Vector object at the same time. On the other hand, an ArrayList object is not synchronized, so more than one thread can view it at the same time.

Resize: When a Vector object is full, it simply gets twice as big. An ArrayList object, on the other hand, gets half as big.

Since Vector is synchronized, it can be slower than ArrayList in apps with only one thread. But Vector may be faster in multithreaded apps where thread safety is important because it has built-in scheduling.



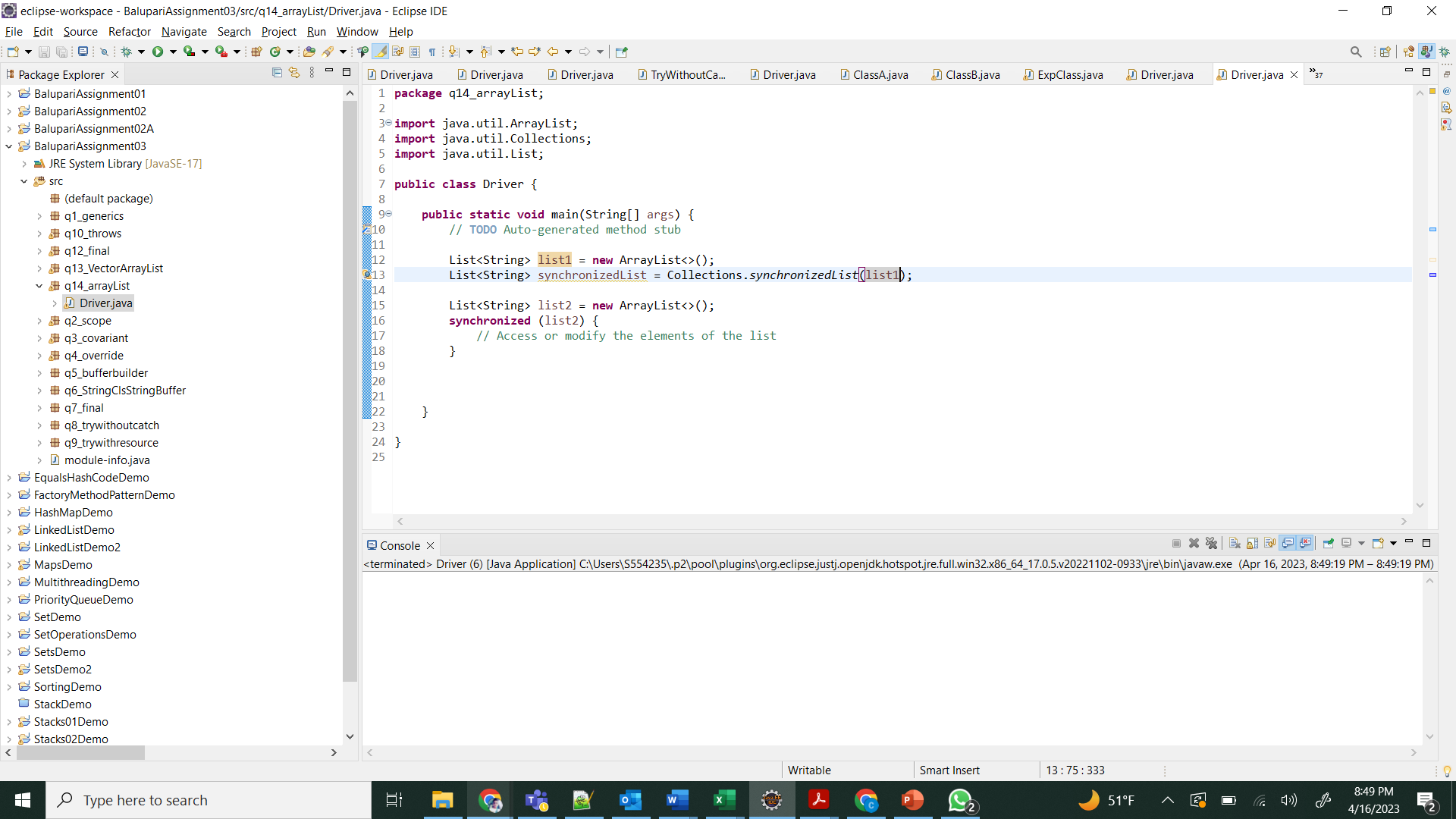
1. What are the different ways to make ArrayList methods synchronized?

ArrayList methods can be made synchronized in different ways:

How to Use Collections.synchronizedList() is a useful method in the Collections class that returns a synchronized (thread-safe) list that is backed by the given list.

Using a synchronized block: Using a synchronized block lets you control who can view an ArrayList object at the same time.

Extending the Vector class: The Vector class implements the List interface in a synced way. You can build on the Vector class and use the parent class's methods to make sure that the kid class's methods are also synchronized. But this isn't the best way to do it because Vector is an old-fashioned collection and ArrayList is better.



1. Difference between Hash table and Hash Map?

HashTable and HashMap are both used to store key-value pairs in Java, but there are some changes between the two:

Synchronization: HashTable is synced, which means that multiple threads can access it at the same time without messing up the data. HashMap, on the other hand, is not synced, so it is not thread-safe by default.

Null keys and values: HashTable doesn't let you have either null keys or values, but HashMap does.

Iterator: The iterator that HashTable gives back is an old iterator, while the iterator that HashMap gives back is a fail-fast iterator.

HashMap is faster than HashTable because it doesn't have to keep track of everything at once. But if you need thread safety, HashTable can be faster than HashMap because synchronization can slow things down.

A screenshot of a computer

Description automatically generated

1. In Java 8, explain how Hasp Map internally works?

Inside HashMap in Java 8, key-value pairs are stored as a collection of Node objects. Each Node object has a key, a value, a hash code, and a link to the next Node in the chain. This is called a "hash table with linked list buckets" or a "hash map with separate chaining."

When a key-value pair is added to the HashMap, the hashCode() method is used to figure out the key's hash code. Then, the hash code is used to figure out where the Node object should go in the list. If there isn't already a Node at that number, a new one is made and put there. If there is already a Node at that index, a linked list of Node objects is made at that index, and the new Node is put to the beginning of the list. This is called "collision resolution," and it's needed because different keys can have the same hash code, which means they'll be kept in the array at the same index.

When a key is looked up in the HashMap, the key's hash code is recalculated and the number in the array is found. If there is no Node at that number, then the key is not in the HashMap. If there is a Node at that number, the linked list of Node objects is moved through until a Node with a matching key is found or the end of the list is reached. If a Node is found with a key that matches, the value for that Node is returned.

In Java 8, "treeification" is also used by HashMap to speed up get() and put() actions on HashMaps with a lot of entries. When a bucket has more than a certain number of entries, it is changed from a linked list to a binary tree, which makes finding and adding items faster.

1. Difference between fail fast and fail-safe iterator?

The difference between a fail-fast iterator and a fail-safe iterator is how they deal with changes to the collection as they go through it.

A fail-fast iterator checks to see if another thread or process has changed the collection, and if so, it throws a ConcurrentModificationException right away. It makes sure that the collection stays in the same state and finds problems like changes being made at the same time. Most Java collection classes, like ArrayList, HashMap, etc., use fail-fast iterators.

A fail-safe iterator, on the other hand, makes a copy of the collection it is iterating over and works on that copy instead of the original collection. This makes sure that the original collection stays the same and that any changes to it don't change the way the repeat works. Most of the time, fail-safe iterators are used in applications where more than one thread can change the same collection at the same time.

A screenshot of a computer

Description automatically generated

1. Can we start the thread twice?

We can't start a thread more than once. When a thread is started, it goes into a state called "running" and starts to run the code in its run() method. A java.lang.IllegalThreadStateException is thrown if we try to start the same thread again.

If we need to run the same code again, we need to make a new copy of the thread and start it. We could also make the run() method run more than once by using a loop or some other control structure.

Graphical user interface, application

Description automatically generated

1. What are the different ways to create a thread in java? Which one is preferred?

In Java, there are two ways to make a thread:

By extending the Thread class, we can make a new class that extends the Thread class and modify the run() method to set the code that the thread will run.

By implementing the Runnable interface: We can make a new class that uses the Runnable interface and pass an instance of this class to the Thread constructor. In the Runnable class, the run() method needs to be set up.

Both ways of doing things are correct, and each has its own pros and cons. It's easier and quicker to extend the Thread class, but we can't then extend any other classes. We can extend other classes and apply multiple interfaces by implementing the Runnable interface. It also gives a better separation of issues by keeping the behavior of the thread separate from the structure of the objects.

In general, it is better to implement the Runnable interface than to extend the Thread class, unless we need to modify some of the Thread methods or we are working with legacy code that already extends the Thread class.

A screenshot of a computer

Description automatically generated

1. What are the different states a thread will go through?

During its life, a Java thread can be in one of several states, which depend on what it is doing at the time and how the operating system plans its execution. These are the states:

When a new thread object is made, it is in a state called "new." The thread hasn't started to move yet.

Runnable: A thread object is in the runnable state when the start() method is called on it. This means that the thread can run, but it might not be running right now because other threads are.

Running: A thread is in the running state when the thread manager picks it to run. In this state, the run() method of the thread is run.

Blocked: A thread is in the blocked state when it is waiting for a monitor lock to be freed. This can happen if the thread tries to synchronize on an object that is already locked by another thread.

Waiting: A thread can be in the "waiting" state while it's waiting for another thread to do something. For instance, a thread can wait for the user to enter something or for a timer to run out.

Timed Waiting: A thread can be in the timed waiting state when it waits for a certain amount of time before moving on. This can happen when the thread waits for a certain condition to be met or when it goes to sleep for a certain amount of time.

Terminated: A thread is in the terminated state when its run() method finishes or when an exception is thrown that isn't being handled.

It's important to remember that the Java program can't see all the states of all threads. The operating system might have more states or changes between states that the Java program doesn't know about.

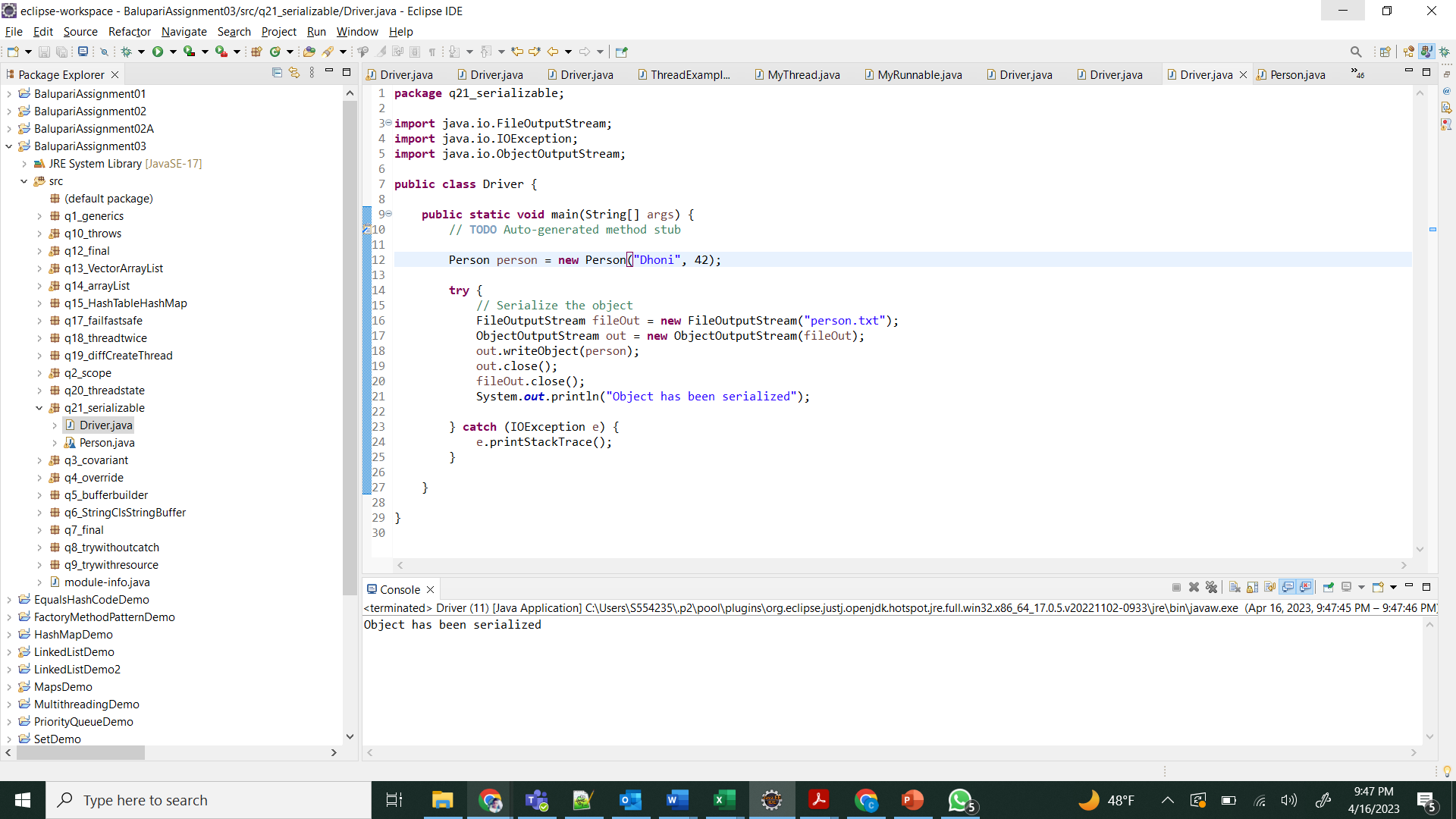
A screenshot of a computer

Description automatically generated

1. What is Serialization? How do we achieve it?

Serialization is the process of turning an object into a stream of bytes that can be saved to a file, sent over a network, or kept in a database. Deserialization is the process of turning the stream of bytes that was turned into an object back into a stream of bytes.

In Java, the Serializable method is used to serialize things. When a class uses the Serializable interface, it shows that its objects can be serialized. A member of the ObjectOutputStream class is used for serialization, and a member of the ObjectInputStream class is used for deserialization.

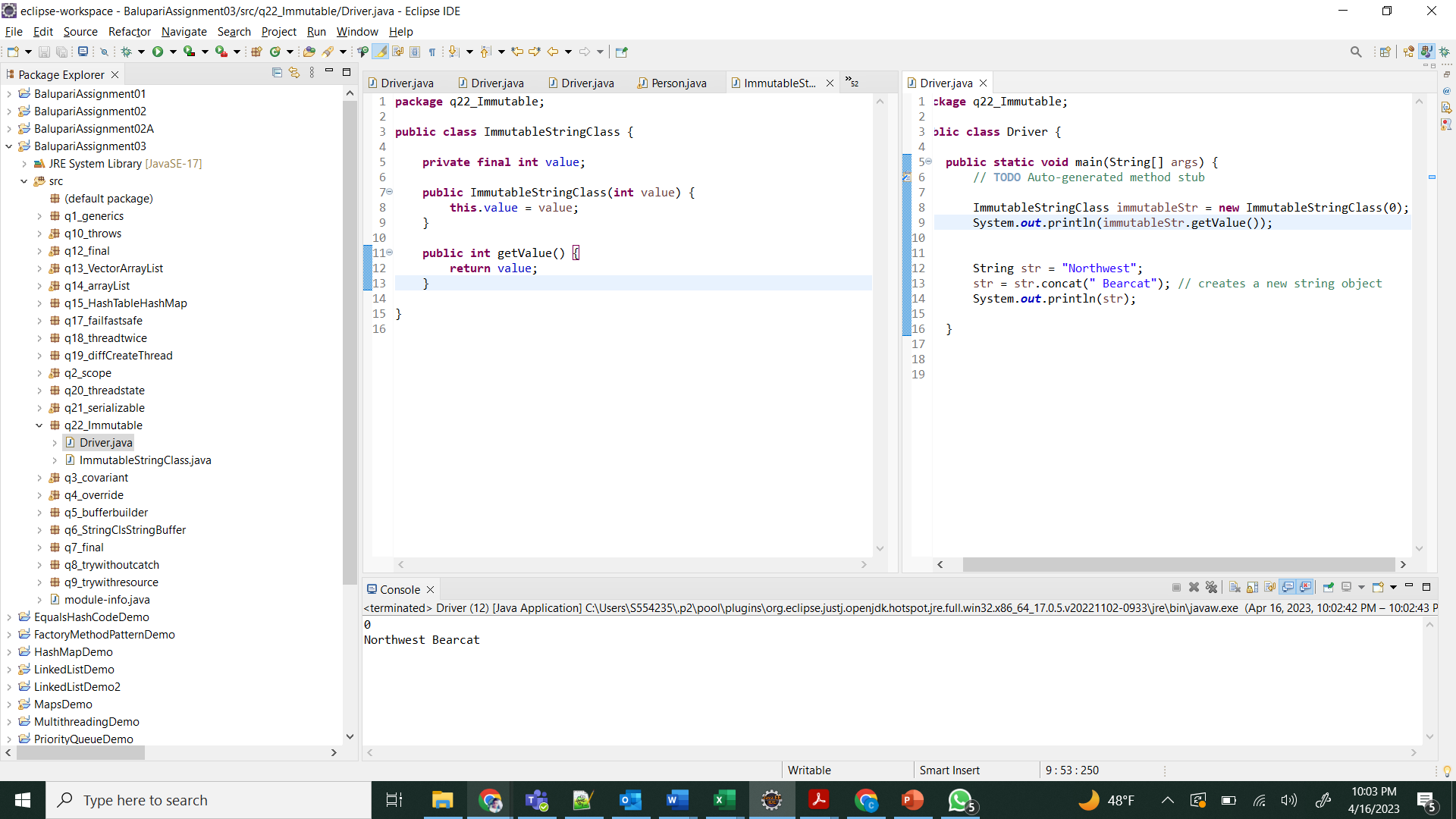


1. What is immutable class? Is String class immutable?

A class is immutable if its instances can't be changed after they've been made. In other words, once a permanent object is made, its state can't be changed.

In Java, String is a class that can't be changed. The value of a String object can't be changed once it's been made. When you do anything to a String, like concatenate or substring, a new String object is made instead of changing the old String object.

This String class feature makes sure that the string data is safe and can't be changed by other threads. It also makes it possible to use texts as keys in hash tables and other data structures.



1. Do immutable classes thread safe? If yes then how?

Yes, by definition, classes that can't be changed are thread-safe. Since an immutable object's state can't be changed after it's been made, multiple threads can access it at the same time without any problems.

A screenshot of a computer

Description automatically generated

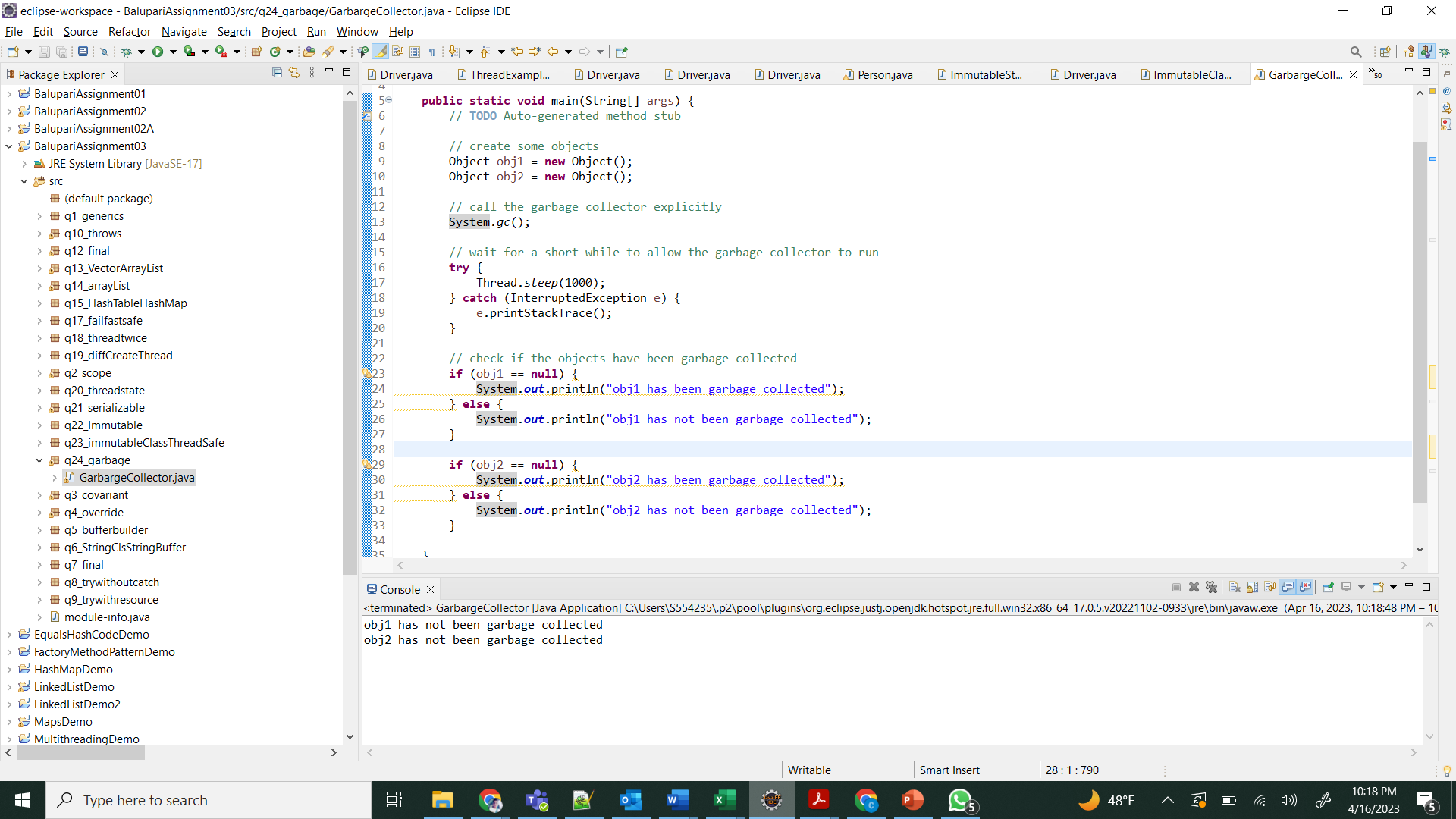
Since the value variable is final and can't be changed after it is set, there is no need to synchronize when viewing it. The getValue() method of an ImmutableClass object can be used by multiple threads at the same time without any problems.

Immutability is a key design concept for making Java thread-safe. By making sure that objects can't be changed after they've been made, you prevent different threads from changing the same object at the same time and causing "race conditions."

1. Can we call the garbage collector explicitly? Will it trigger the garbage collector?

Yes, we can use the System.gc() method to tell the trash collector to do its job. But there's no promise that the garbage collector will come right away. The garbage collector is set up to run immediately when the JVM decides that memory needs to be freed up. This means that most of the time, you don't need to call the garbage collector directly.

When System.gc() is called, it just tells the garbage collector that it needs to run. At that point, the trash collector may or may not decide to run, and it may not pick up all objects that are supposed to be picked up. So, it's usually not a good idea to call the garbage collection directly in production code, since it can slow things down unnecessarily.



1. What are Java 8 features? Explain all of them with examples?

The Java language got a lot of new features and improvements with Java 8. Here are a few of the most important ones:

Lambda Expressions: Lambda expressions are a new way to write code for functional computing that is short and easy to understand. They let you pass behavior to a method as an input or make a function that doesn't have a name.

Stream API: The Stream API is a fluent API that lets us do complicated data processing tasks on collections. It gives a good way to handle groups of objects, making it easy to run tasks in parallel.

Default Methods: Interfaces can have application code when they have default methods. They make it possible for interfaces to change without breaking the code that makes them work.

Method references are a shorthand way to write lambda statements that call only one method.

Date/Time API: The new Date/Time API makes working with dates and times easier and less likely to go wrong.

Optional Class: The Optional class gives you a more clear and safe way to deal with null values.

These are some of the most important things that Java 8 added. There are also a number of other features and improvements, such as CompletableFuture, repeatable comments, and more.

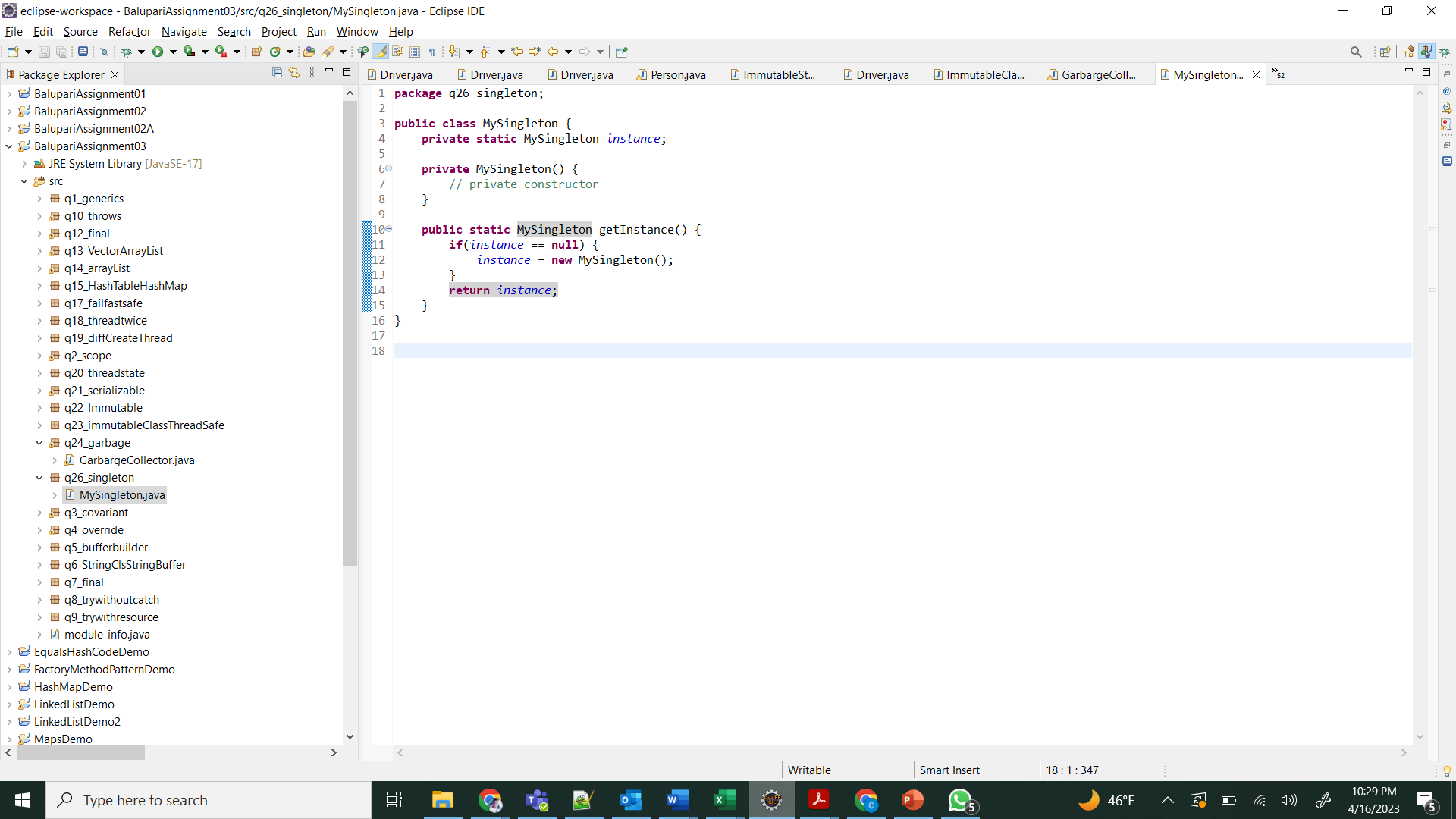
1. How to make a pure singleton?

To make a clean singleton, we need to make sure that only one instance of the class can be made and that it can be accessed everywhere. One way to do this is as follows:

Make the constructor secret to stop the new operator from being used to create objects.

Make a secret variable that is static and will hold the instance of the class.

Make a public static method that returns the class's instance. This method should make the object if it hasn't been made yet, and if it already exists, it should return it.



1. How to make a singleton synchronized?

To link a singleton, you need to make sure that only one thread can access it at a time. Adding the word "synchronized" to the getInstance() method is one way to do this.

If the getInstance() method is synchronized, other threads may have to wait for the synchronized block to finish before they can access the instance. This can hurt speed. As a result, the getInstance() method should only be synchronized if thread safety is truly necessary. If your app doesn't need strict thread safety, you can make a singleton copy without synchronization by using the double-checked locking pattern or the static initialization pattern.

A screenshot of a computer

Description automatically generated

Git Repo link: <https://github.com/bkrishnasai1998/Patterns-BalupariAssignment03.git>

**Submission:**  Change the document name as ***Lastname*Assignment03** where *Lastname* is your Last name and submit.