Assignment 1

ANSWER THE FOLLOWING SHORT QUESTIONS (EACH QUESTION CARRIES THREE MARKS) (3*5=15)

1) Write about benefits of OOP?
2) List out and explain about character and byte streams clases?
3) Describe collection classes along with a suitable example?
4) List out event lister interfaces and write about event delegation model

5) explain process of reading and writing into files with an example

Assignment 2

ANSWER THE FOLLOWING QUESTIONS (EACH QUESTIONS CARRIES FIVE MARKS) (3*5=15)

1) Explain about different types of control statements with an example
2) Write about multithreading programming and explain how synchronization is achieved
3) What is the usage and purpose of string tokenization with an example
4) Write about awt controls with an example
5) Write about java network programming with an example

Assignment 1

1.Write about benefits of OOP?

Object-Oriented Programming or OOPs refers to languages that use objects in programming, they use objects as a primary source to implement what is to happen in the code. Objects are seen by the viewer or user, performing tasks assigned by you. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc. in programming

Object means a real-world entity such as a pen, chair, table, computer, watch, etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance

Benefits of OOPs in Java

Let's understand these one by one in detail!

1. Modularity

- OOP promotes the division of a software program into distinct modules or classes, each representing a specific component or functionality.
- This modularity makes the code easier to manage, maintain, and understand.

2. Reusability

- OOP encourages code reuse through inheritance and composition.
- By creating new classes from existing ones, developers can leverage pre-existing code, reducing redundancy and effort.

3. Encapsulation

- Encapsulation hides the internal implementation details of classes and exposes only the necessary interfaces.
- This leads to better data protection and reduces the likelihood of unintended interference or misuse of data.

4. Inheritance

- Inheritance allows new classes to inherit properties and methods from existing classes.
- This promotes code reuse, establishes a natural hierarchy, and enables polymorphism.

5. Polymorphism

- Polymorphism allows methods to behave differently based on the object invoking them, enabling flexibility and scalability in code.

- It simplifies code maintenance and enhances the ability to extend the system with new functionalities without modifying existing code.

6. Abstraction

- Abstraction simplifies complex systems by modeling classes appropriate to the problem domain.
- It allows focusing on the essential features of an object, reducing complexity and improving code clarity.

7. Improved Maintainability

- OOP's modularity and encapsulation make updating, modifying, and debugging code easier.
- Changes in one part of the system have minimal impact on other parts, enhancing maintainability.

8. Ease of Troubleshooting

- OOP simplifies the process of debugging and troubleshooting by isolating functionality into separate classes.
- Problems can be localized and resolved more efficiently.

9. Real-World Modeling

- OOP helps in the modeling of real-world entities and relationships in software.
- This makes the code more intuitive and aligned with human understanding of the problem domain.

10. Enhanced Collaboration

- OOP supports collaborative development by enabling multiple developers to work on different classes or modules simultaneously.
- Well-defined interfaces and encapsulation improve teamwork and integration.

11. Extensibility

- OOP's principles make it easier to extend and scale software systems.
- New features and functionalities can be added with minimal disruption to existing code.

2.List out and explain about character and byte streams clases?

Character Stream and Byte Stream in Java

In Java the streams are used for input and output operations by allowing data to be read from or written to a source or destination.

Java offers two types of streams:

1. character streams
2. byte streams.

These streams can be different in how they are handling data and the type of data they are handling.

1. Character Streams:

Character streams are designed to address character based records, which includes textual records inclusive of letters, digits, symbols, and other characters. These streams are represented by way of training that quit with the phrase "Reader" or "Writer" of their names, inclusive of FileReader, BufferedReader, FileWriter, and BufferedWriter.

Character streams offer a convenient manner to read and write textual content-primarily based information due to the fact they mechanically manage character encoding and decoding. They convert the individual statistics to and from the underlying byte circulation the usage of a particular individual encoding, such as UTF-eight or ASCII.It makes person streams suitable for operating with textual content files, analyzing and writing strings, and processing human-readable statistics.

2. Byte Streams:

Byte streams are designed to deal with raw binary data, which includes all kinds of data, including characters, pictues, audio, and video. These streams are represented through cclasses that cease with the word "InputStream" or "OutputStream" of their names,along with FileInputStream,BufferedInputStream, FileOutputStream and BufferedOutputStream.

Byte streams offer a low-stage interface for studying and writing character bytes or blocks of bytes. They are normally used for coping with non-textual statistics, studying and writing files of their binary form, and running with network sockets. Byte streams don't perform any individual encoding or deciphering. They treat the data as a sequence of bytes and don't interpret it as characters.

Here are the some of the differences listed:

Aspect	Character Streams	Byte Streams
Data Handling	Handle character-based data	Handle raw binary data

Representation	Classes end with "Reader" or "Writer"	Classes end with "InputStream" or "OutputStream"
Suitable for	Textual data, strings, human-readable info	Non-textual data, binary files, multimedia
Character Encoding	Automatic encoding and decoding	No encoding or decoding
Text vs non-Text data	Text-based data, strings	Binary data, images, audio, video
Performance	Additional conversion may impact performance	Efficient for handling large binary data
Handle Large Text Files	May impact performance due to encoding	Efficient, no encoding overhead
String Operations	Convenient methods for string operations	Not specifically designed for string operations
Convenience Methods	Higher-level abstractions for text data	Low-level interface for byte data
Reading Line by Line	Convenient methods for reading lines	Byte-oriented, no built-in line-reading methods
File Handling	Read/write text files	Read/write binary files
Network Communication	Sending/receiving text data	Sending/receiving binary data
Handling Images/Audio/Video	Not designed for handling binary data directly	Suitable for handling binary multimedia data
Text Encoding	Supports various character encodings	No specific text encoding support

3.Describe collection classes along with a suitable example?

Collections Class in Java

Last Updated : 07 May, 2023

- **Collections** class in Java is one of the utility classes in Java Collections Framework. The java.util package contains the Collections class in Java. Java Collections class is used with the static methods that operate on the collections or return the collection. All the methods of this class throw the **NullPointerException** if the collection or object passed to the methods is null.

Collection Class declaration

The syntax of the Collection class declaration is mentioned below:

```
public class Collections extends Object
```

Remember: Object is the parent class of all the classes.

Java Collection Class

Collection Framework contains both classes and interfaces. Although both seem the same but there are certain differences between Collection classes and the Collection framework. There are some classes in Java as mentioned below:

1. ArrayList

ArrayList is a class implemented using a list interface, in that provides the functionality of a dynamic array where the size of the array is not fixed.

Syntax:

```
ArrayList<_type_> var_name = new ArrayList<_type_>();
```

2. Vector

Vector is a Part of the collection class that implements a dynamic array that can grow or shrink its size as required.

Syntax:

```
public class Vector<E> extends AbstractList<E> implements List<E>,
RandomAccess,

Cloneable, Serializable
```

3. Stack

Stack is a part of Java collection class that models and implements a Stack data structure. It is based on the basic principle of last-in-first-out(LIFO) .
Syntax:

```
public class Stack<E> extends Vector<E>
```

4. LinkedList

LinkedList class is an implementation of the LinkedList data structure. It can store the elements that are not stored in contiguous locations and every element is a separate object with a different data part and different address part.
Syntax:

```
LinkedList name = new LinkedList();
```

5. HashSet

HashSet is implemented using the Hashtable data structure. It offers constant time performance for the performing operations like add, remove, contains, and size.
Syntax:

```
public class HashSet<E> extends AbstractSet<E> implements Set<E>,
Cloneable, Serializable
```

6. LinkedHashSet

LinkedHashSet is an ordered version of HashSet that maintains a doubly-linked List across all elements.
Syntax:

```
public class LinkedHashSet<E> extends HashSet<E> implements Set<E>,
Cloneable, Serializable
```

7. TreeSet

TreeSet class is implementations of the SortedSet interface in Java that uses a Tree for storage. The ordering of the elements is maintained by a set using their natural ordering whether an explicit comparator is provided or not.
Syntax:

```
TreeSet t = new TreeSet(Collection col);
```

8. PriorityQueue

The PriorityQueue is based on the priority heap. The elements of the priority queue are ordered according to the natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.
Syntax:

```
public class PriorityQueue<E> extends AbstractQueue<E> implements
Serializable
```

9. ArrayDeque

The ArrayDeque class in Java is an implementation of the Deque interface that uses a resizable array to store its elements. The ArrayDeque class provides constant-time performance for inserting and removing elements from both ends.
Syntax:

```
public class ArrayDeque<E> extends

AbstractCollection<E> implements Deque<E>, Cloneable,Serializable
```

10. HashMap

HashMap Class is similar to HashTable but the data unsynchronized. It stores the data in (Key, Value) pairs, and you can access them by an index of another type.
Syntax:

```
public class HashMap<K,V> extends AbstractMap<K,V>

implements Map<K,V>, Cloneable, Serializable
```

11. EnumMap

EnumMap extends AbstractMap and implements the Map interface in Java.
Syntax:

```
public class EnumMap<K extends Enum<K>,V> extends

AbstractMap<K,V> implements Serializable, Cloneable
```

12. AbstractMap

The AbstractMap class is a part of the Java Collection Framework. It implements the Map interface to provide a structure to it, by doing so it makes the further implementations easier.
Syntax:

```
public abstract class AbstractMap<K,V> extends Object, implements
Map<K,V>
```

13. TreeMap

A TreeMap is implemented using a Red-Black tree.TreeMap provides an ordered collection of key-value pairs, where the keys are ordered based on their natural order or a custom Comparator passed to the constructor.

```
SortedMap m = Collections.synchronizedSortedMap(new TreeMap(...));
```

4.List out event lister interfaces and write about event delegation model

Delegation Event Model in Java

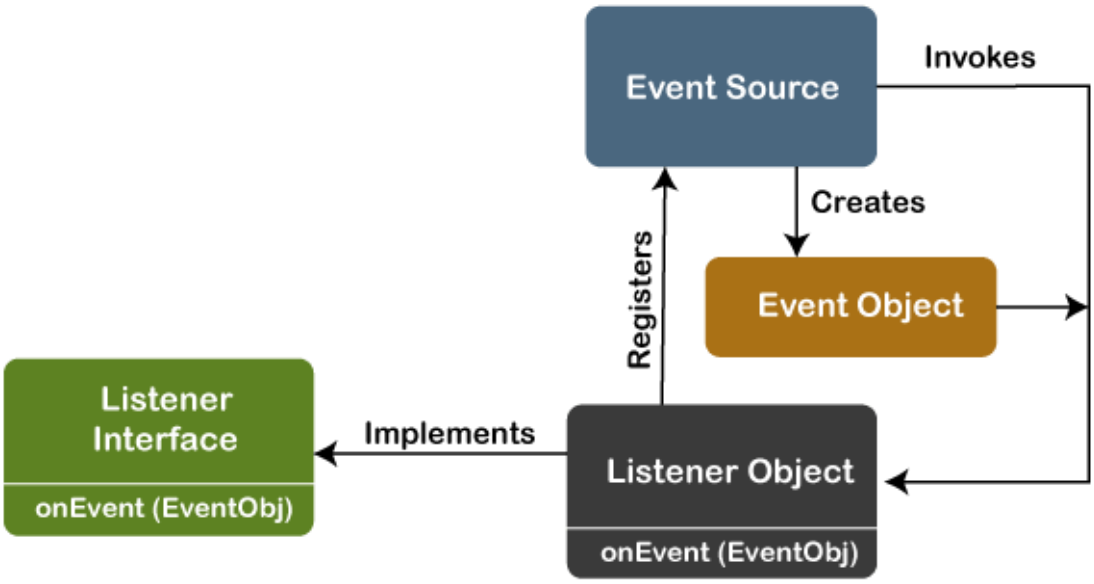
The Delegation Event model is defined to handle events in GUI programming languages. The GUI stands for Graphical User Interface, where a user graphically/visually interacts with the system.

The GUI programming is inherently event-driven; whenever a user initiates an activity such as a mouse activity, clicks, scrolling, etc., each is known as an event that is mapped to a code to respond to functionality to the user. This is known as event handling.

In this section, we will discuss event processing and how to implement the delegation event model in Java. We will also discuss the different components of an Event Model.

Event Processing in Java

Java support event processing since Java 1.0. It provides support for AWT (Abstract Window Toolkit), which is an API used to develop the Desktop application. In Java 1.0, the AWT was based on inheritance. To catch and process GUI events for a program, it should hold subclass GUI components and override action() or handleEvent() methods. The below image demonstrates the event processing.



But, the modern approach for event processing is based on the Delegation Model. It defines a standard and compatible mechanism to generate and process events. In this model, a source generates an event and forwards it to one or more listeners. The listener waits until it receives an event. Once it receives the event, it is processed by the listener and returns it. The UI elements are able to delegate the processing of an event to a separate function.

The key advantage of the Delegation Event Model is that the application logic is completely separated from the interface logic.

In this model, the listener must be connected with a source to receive the event notifications. Thus, the events will only be received by the listeners who wish to receive them. So, this approach is more convenient than the inheritance-based event model (in Java 1.0).

In the older model, an event was propagated up the containment until a component was handled. This needed components to receive events that were not processed, and it took lots of time. The Delegation Event model overcame this issue.

Basically, an Event Model is based on the following three components:

- Events
- Events Sources
- Events Listeners

Events

The Events are the objects that define state change in a source. An event can be generated as a reaction of a user while interacting with GUI elements. Some of the event generation activities are moving the mouse pointer, clicking on a button, pressing the keyboard key, selecting an item from the list, and so on. We can also consider many other user operations as events.

Event Listeners

An event listener is an object that is invoked when an event triggers. The listeners require two things; first, it must be registered with a source; however, it can be registered with several resources to receive notification about the events. Second, it must implement the methods to receive and process the received notifications.

The methods that deal with the events are defined in a set of interfaces. These interfaces can be found in the java.awt.event package.

For example, the **MouseMotionListener** interface provides two methods when the mouse is dragged and moved. Any object can receive and process these events if it implements the MouseMotionListener interface.

5.explain process of reading and writing into files with an example

File Operations in Java

In Java, a **File** is an abstract data type. A named location used to store related information is known as a **File**. There are several **File Operations** like **creating a new File, getting information about File, writing into a File, reading from a File** and **deleting a File**.

Before understanding the File operations, it is required that we should have knowledge of **Stream** and **File methods**.

File Operations

We can perform the following operation on a file:

- Create a File
- Get File Information
- Write to a File
- Read from a File
- Delete a File

Create a File

Create a File operation is performed to create a new file. We use the **createNewFile()** method of file. The **createNewFile()** method returns true when it successfully creates a new file and returns false when the file already exists.

Let's take an example of creating a file to understand how we can use the **createNewFile()** method to perform this operation.

Get File Information

The operation is performed to get the file information. We use several methods to get the information about the file like name, absolute path, is readable, is writable and length.

Write to a File

The next operation which we can perform on a file is **"writing into a file"**. In order to write data into a file, we will use the **FileWriter** class and its **write()** method together. We need to close the stream using the **close()** method to retrieve the allocated resources.

WriteToFile.java

```
1. // Importing the FileWriter class
2. import java.io.FileWriter;
3.
4. // Importing the IOException class for handling errors
5. import java.io.IOException;
6.
7. class WriteToFile {
8.     public static void main(String[] args) {
9.
10.    try {
11.        FileWriter fwrite = new FileWriter("D:FileOperationExample.txt");
12.        // writing the content into the FileOperationExample.txt file
13.        fwrite.write("A named location used to store related information is referred to as
    a File.");
14.
15.        // Closing the stream
16.        fwrite.close();
```

```
17.         System.out.println("Content is successfully wrote to the file.");
18.     } catch (IOException e) {
19.         System.out.println("Unexpected error occurred");
20.         e.printStackTrace();
21.         }
22.     }
23. }
```

Read from a File

The next operation which we can perform on a file is **"read from a file"**. In order to write data into a file, we will use the **Scanner** class. Here, we need to close the stream using the **close()** method. We will create an instance of the Scanner class and use the **hasNextLine()** method **nextLine()** method to get data from the file.

ReadFromFile.java

```
1.  // Importing the File class
2.  import java.io.File;
3.  // Importing FileNotFoundException class for handling errors
4.  import java.io.FileNotFoundException;
5.  // Importing the Scanner class for reading text files
6.  import java.util.Scanner;
7.
8.  class ReadFromFile {
9.      public static void main(String[] args) {
10.         try {
11.             // Create f1 object of the file to read data
12.             File f1 = new File("D:FileOperationExample.txt");
13.             Scanner dataReader = new Scanner(f1);
14.             while (dataReader.hasNextLine()) {
15.                 String fileData = dataReader.nextLine();
16.                 System.out.println(fileData);
17.             }
18.             dataReader.close();
19.         } catch (FileNotFoundException exception) {
20.             System.out.println("Unexcpected error occurred!");
21.             exception.printStackTrace();
```

```
22.         }
23.     }
24. }
```

Delete a File

The next operation which we can perform on a file is **"deleting a file"**. In order to delete a file, we will use the **delete()** method of the file. We don't need to close the stream using the **close()** method because for deleting a file, we neither use the FileWriter class nor the Scanner class.

Assignment 2

1. Explain about different types of control statements with an example

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java provides three types of control flow statements.

1. Decision Making statements
    - if statements
    - switch statement
2. Loop statements
    - do while loop
    - while loop
    - for loop
    - for-each loop
3. Jump statements
    - break statement
    - continue statement

Decision-Making statements:

As the name suggests, decision-making statements decide which statement to execute and when. Decision-making statements evaluate the Boolean expression and

control the program flow depending upon the result of the condition provided. There are two types of decision-making statements in Java, i.e., If statement and switch statement.

1) If Statement:

In Java, the "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false. In Java, there are four types of if-statements given below.

1. Simple if statement
2. if-else statement
3. if-else-if ladder
4. Nested if-statement

1) Simple if statement:

It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

Syntax of if statement is given below.

```
1. if(condition) {
2. statement 1; //executes when condition is true
3. }
```

2) if-else statement

The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

Syntax:

```
1. if(condition) {
2. statement 1; //executes when condition is true
3. }
4. else{
5. statement 2; //executes when condition is false
6. }
```

3) if-else-if ladder:

The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. We can also define an else statement at the end of the chain.

Syntax of if-else-if statement is given below.

```
1. if(condition 1) {
2. statement 1; //executes when condition 1 is true
3. }
4. else if(condition 2) {
5. statement 2; //executes when condition 2 is true
6. }
7. else {
8. statement 2; //executes when all the conditions are false
9. }
```

4. Nested if-statement

In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.

Syntax of Nested if-statement is given below.

```
1. if(condition 1) {
2. statement 1; //executes when condition 1 is true
3. if(condition 2) {
4. statement 2; //executes when condition 2 is true
5. }
6. else{
7. statement 2; //executes when condition 2 is false
8. }
9. }
```

Switch Statement:

In Java, Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

```
switch (expression){
    case value1:
     statement1;
     break;
     ……..
    case valueN:
     statementN;
     break;
    default:
     default statement;
}
```

Loop Statements

In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order. The execution of the set of instructions depends upon a particular condition.

In Java, we have three types of loops that execute similarly. However, there are differences in their syntax and condition checking time.

1. for loop
2. while loop
3. do-while loop

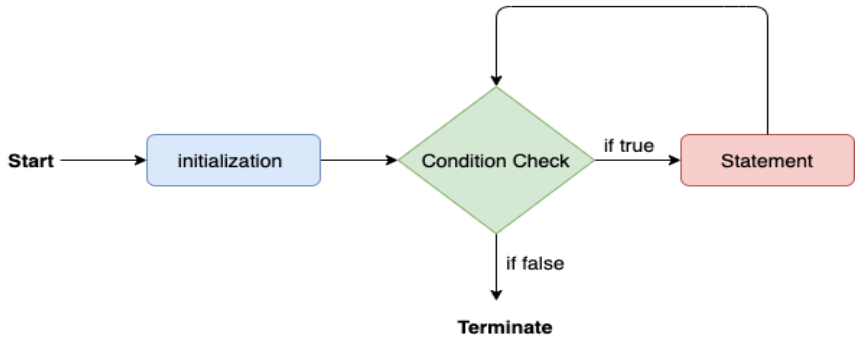
Let's understand the loop statements one by one.

Java for loop

In Java, for loop is similar to C and C++. It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code. We use the for loop only when we exactly know the number of times, we want to execute the block of code.

1. **for**(initialization, condition, increment/decrement) {
2. //block of statements
3. }

The flow chart for the for-loop is given below.



Java while loop

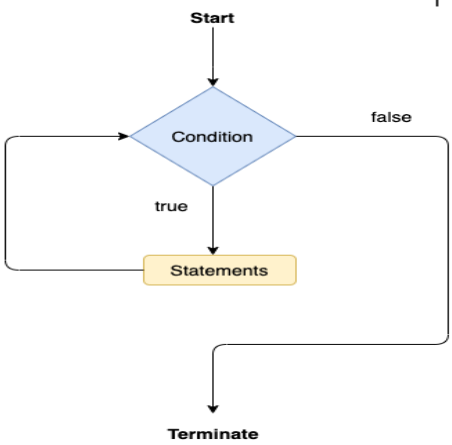
The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop. Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop.

It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

The syntax of the while loop is given below.

1. **while**(condition){
2. //looping statements
3. }

4. The flow chart for the while loop is given in the following image.



5.

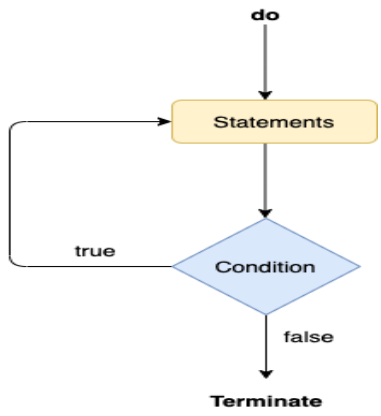
Java do-while loop

The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.

It is also known as the exit-controlled loop since the condition is not checked in advance. The syntax of the do-while loop is given below.

```
1. do
2. {
3. //statements
4. } while (condition);
```

The flow chart of the do-while loop is given in the following image.



Jump Statements

Jump statements are used to transfer the control of the program to the specific statements. In other words, jump statements transfer the execution control to the other part of the program. There are two types of jump statements in Java, i.e., break and continue.

Java break statement

As the name suggests, the break statement is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop.

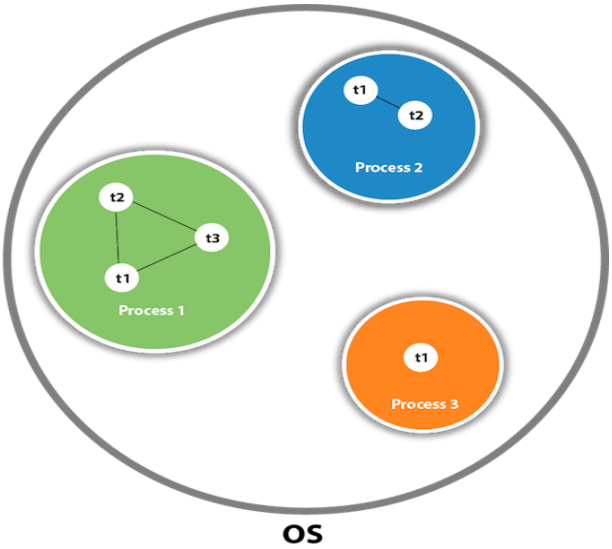
Java continue statement

Unlike break statement, the continue statement doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

2.Write about multithreading programming and explain how synchronization is achieved

A thread is a lightweight subprocess, the smallest unit of processing. It is a separate path of execution.

Threads are independent. If there occurs exception in one thread, it doesn't affect other threads. It uses a shared memory area.



As shown in the above figure, a thread is executed inside the process. There is context-switching between the threads. There can be multiple processes inside the OS, and one process can have multiple threads.

Multithreading in Java

Multithreading in Java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation, etc.

Advantages of Java Multithreading

1) It **doesn't block the user** because threads are independent and you can perform multiple operations at the same time.

2) You **can perform many operations together, so it saves time**.

3) Threads are **independent**, so it doesn't affect other threads if an exception occurs in a single thread.

Synchronization in Java

Synchronization in Java is the capability to control the access of multiple threads to any shared resource.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Why use Synchronization?

The synchronization is mainly used to

1. To prevent thread interference.
2. To prevent consistency problem.

Types of Synchronization

There are two types of synchronization

1. Process Synchronization
2. Thread Synchronization

Thread Synchronization

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive

    1. Synchronized method.
    2. Synchronized block.
    3. Static synchronization.
2. Cooperation (Inter-thread communication in java)

Mutual Exclusive

Mutual Exclusive helps keep threads from interfering with one another while sharing data. It can be achieved by using the following three ways:

1. By Using Synchronized Method
2. By Using Synchronized Block
3. By Using Static Synchronization

Java Synchronized Method

If you declare any method as synchronized, it is known as synchronized method.

Synchronized method is used to lock an object for any shared resource.

When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

Synchronized Block in Java

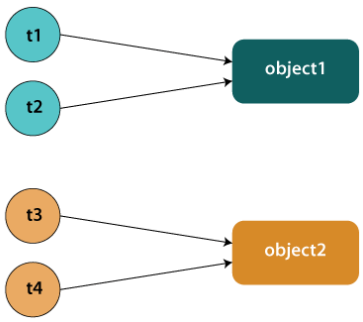
Synchronized block can be used to perform synchronization on any specific resource of the method.

Suppose we have 50 lines of code in our method, but we want to synchronize only 5 lines, in such cases, we can use synchronized block.

If we put all the codes of the method in the synchronized block, it will work same as the synchronized method.

Static Synchronization

If you make any static method as synchronized, the lock will be on the class not on object.



Problem without static synchronization

Suppose there are two objects of a shared class (e.g. Table) named object1 and object2. In case of synchronized method and synchronized block there cannot be interference between t1 and t2 or t3 and t4 because t1 and t2 both refers to a common object that have a single lock. But there can be interference between t1 and t3 or t2 and t4 because t1 acquires another lock and t3 acquires another lock. We don't want interference between t1 and t3 or t2 and t4. Static synchronization solves this problem.

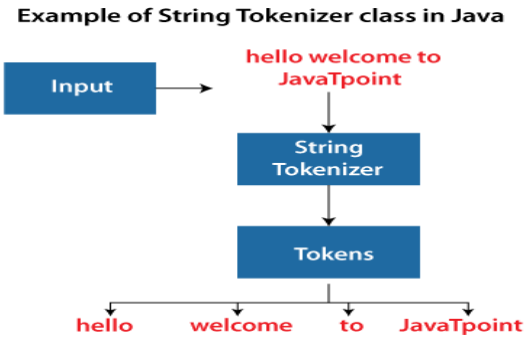
3.What is the usage and purpose of string tokenization with an example

StringTokenizer in Java

The **java.util.StringTokenizer** class allows you to break a String into tokens. It is simple way to break a String. It is a legacy class of Java.

It doesn't provide the facility to differentiate numbers, quoted strings, identifiers etc. like StreamTokenizer class. We will discuss about the StreamTokenizer class in I/O chapter.

In the StringTokenizer class, the delimiters can be provided at the time of creation or one by one to the tokens.



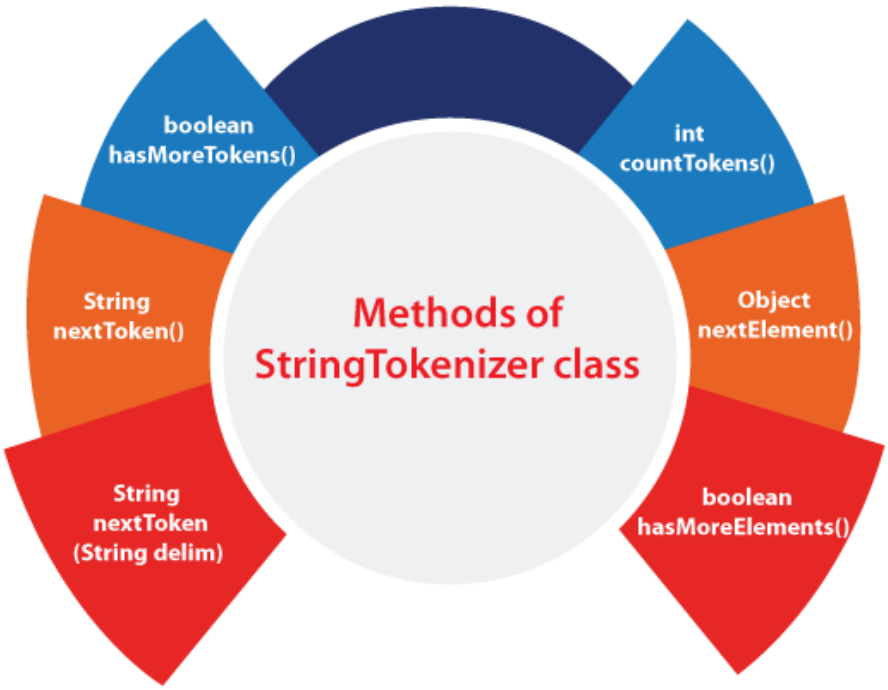
Constructors of the StringTokenizer Class

There are 3 constructors defined in the StringTokenizer class.

Constructor	Description
StringTokenizer(String str)	It creates StringTokenizer with specified string.
StringTokenizer(String str, String delim)	It creates StringTokenizer with specified string and delimiter.
StringTokenizer(String str, String delim, boolean returnValue)	It creates StringTokenizer with specified string, delimiter and returnValue. If return value is true, delimiter characters are considered to be tokens. If it is false, delimiter characters serve to separate tokens.

Methods of the StringTokenizer Class

The six useful methods of the StringTokenizer class are as follows:



Methods	Description
boolean hasMoreTokens()	It checks if there is more tokens available.
String nextToken()	It returns the next token from the StringTokenizer object.
String nextToken(String delim)	It returns the next token based on the delimiter.
boolean hasMoreElements()	It is the same as hasMoreTokens() method.
Object nextElement()	It is the same as nextToken() but its return type is Object.
int countTokens()	It returns the total number of tokens.

Example of StringTokenizer Class

Let's see an example of the StringTokenizer class that tokenizes a string "my name is khan" on the basis of whitespace.

Simple.java

```
1. import java.util.StringTokenizer;
2. public class Simple{
3.  public static void main(String args[]){
4.    StringTokenizer st = new StringTokenizer("my name is khan"," ");
5.      while (st.hasMoreTokens()) {
6.          System.out.println(st.nextToken());
7.      }
8.    }
9. }
```

Output:

```
my
name
is
khan
```

1) 4. Write about awt controls with an example

Java AWT (Abstract Window Toolkit) is *an API to develop Graphical User Interface (GUI) or windows-based applications* in Java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).

The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

The AWT tutorial will help the user to understand Java GUI programming in simple and easy steps.

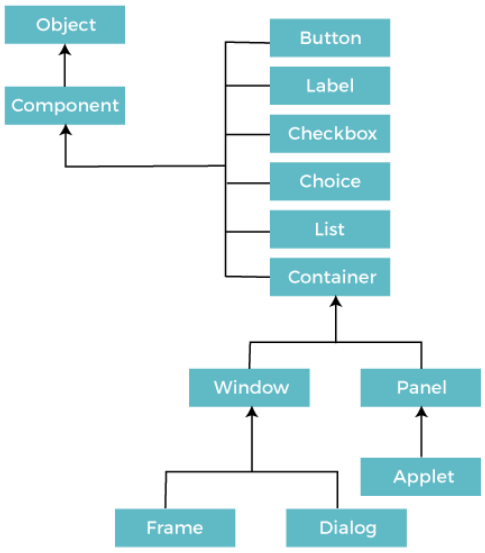
Java AWT calls the native platform calls the native platform (operating systems) subroutine for creating API components like TextField, ChechBox, button, etc.

For example, an AWT GUI with components like TextField, label and button will have different look and feel for the different platforms like Windows, MAC OS, and Unix. The reason for this is the platforms have different view for their native components and AWT directly calls the native subroutine that creates those components.

In simple words, an AWT application will look like a windows application in Windows OS whereas it will look like a Mac application in the MAC OS.

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



Components

All the elements like the button, text fields, scroll bars, etc. are called components. In Java AWT, there are classes for each component as shown in above diagram. In order to place every component in a particular position on a screen, we need to add them to a container.

Container

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as **Frame, Dialog** and **Panel**.

It is basically a screen where the where the components are placed at their specific locations. Thus it contains and controls the layout of components.

Window

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window. We need to create an instance of Window class to create this container.

Panel

The Panel is the container that doesn't contain title bar, border or menu bar. It is generic container for holding the components. It can have other components like button, text field etc. An instance of Panel class creates a container, in which we can add components.

Frame

The Frame is the container that contain title bar and border and can have menu bars. It can have other components like button, text field, scrollbar etc. Frame is most widely used container while developing an AWT application.

Useful Methods of Component Class

Method	Description
public void add(Component c)	Inserts a component on this component.

public void setSize(int width,int height)	Sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	Defines the layout manager for the component.
public void setVisible(boolean status)	Changes the visibility of the component, by default false.

Java AWT Example

To create simple AWT example, you need a frame. There are two ways to create a GUI using Frame in AWT.

1. By extending Frame class (**inheritance**)
2. By creating the object of Frame class (**association**)

AWT Example by Inheritance

Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

AWTExample1.java

```
1. // importing Java AWT class
2. import java.awt.*;
3.
4. // extending Frame class to our class AWTExample1
5. public class AWTExample1 extends Frame {
6.
7.   // initializing using constructor
8.   AWTExample1() {
9.
10.    // creating a button
11.    Button b = new Button("Click Me!!");
12.
13.    // setting button position on screen
14.    b.setBounds(30,100,80,30);
15.
16.    // adding button into frame
```

```
17.     add(b);
18.
19.     // frame size 300 width and 300 height
20.     setSize(300,300);
21.
22.     // setting the title of Frame
23.     setTitle("This is our basic AWT example");
24.
25.     // no layout manager
26.     setLayout(null);
27.
28.     // now frame will be visible, by default it is not visible
29.     setVisible(true);
30. }
31.
32. // main method
33. public static void main(String args[]) {
34.
35. // creating instance of Frame class
36. AWTExample1 f = new AWTExample1();
37.
38. }
39.
40. }
```

5. Write about java network programming with an example

Java Networking

- When computing devices such as laptops, desktops, servers, smartphones, and tablets and an eternally-expanding arrangement of IoT gadgets such as cameras, door locks, doorbells, refrigerators, audio/visual systems, thermostats, and various sensors are sharing information and data with each other is known as networking.

  In simple words, the term network programming or networking associates with writing programs that can be executed over various computer devices, in which all

the devices are connected to each other to share resources using a network. Here, we are going to discuss ***Java Networking***.

- What is Java Networking?
- Common Network Protocols
- Java Network Terminology
- Java Networking Classes
- Java Networking Interfaces
- Socket Programming
- Inet Address
- URL Class

What is Java Networking?

Networking supplements a lot of power to simple programs. With networks, a single program can regain information stored in millions of computers positioned anywhere in the world. Java is the leading programming language composed from scratch with networking in mind. Java Networking is a notion of combining two or more computing devices together to share resources.

All the Java program communications over the network are done at the application layer. The **java.net** package of the J2SE APIs comprises various classes and interfaces that execute the low-level communication features, enabling the user to formulate programs that focus on resolving the problem.

Common Network Protocols

As stated earlier, the **java.net** package of the Java programming language includes various classes and interfaces that provide an easy-to-use means to access network resources. Other than classes and interfaces, the **java.net** package also provides support for the two well-known network protocols. These are:

1. **Transmission Control Protocol (TCP) –** TCP or Transmission Control Protocol allows secure communication between different applications. TCP is a connection-oriented protocol which means that once a connection is established, data can be transmitted in two directions.
2. **User Datagram Protocol (UDP) –** UDP or User Datagram Protocol is a connection-less protocol that allows data packets to be transmitted between different applications.

Java Networking Terminology

In Java Networking, many terminologies are used frequently. These widely used Java Networking Terminologies are given as follows:

1. **IP Address –** An IP address is a unique address that distinguishes a device on the internet or a local network. IP stands for "Internet Protocol." It comprises a set of rules governing the format of data sent via the internet or local network.
   - Example – 192.168.0.1

2. **Port Number –** A port number is a method to recognize a particular process connecting internet or other network information when it reaches a server. The port number is used to identify different applications uniquely. The port number behaves as a communication endpoint among applications.
3. **Protocol –** A network protocol is an organized set of commands that define how data is transmitted between different devices in the same network. Network protocols are the reason through which a user can easily communicate with people all over the world
4. **MAC Address –** MAC address stands for Media Access Control address. It is a bizarre identifier that is allocated to a NIC (Network Interface Controller/ Card). It contains a 48 bit or 64-bit address, which is combined with the network adapter.
5. **Socket –** A socket is one endpoint of a two-way communication connection between the two applications running on the network. The socket mechanism presents a method of inter-process communication (IPC) by setting named contact points between which the communication occurs.
6. **Connection-oriented and connection-less protocol –** In a connection-oriented service, the user must establish a connection before starting the communication. Java Networking classes

Java Networking Classes

The **java.net** package of the Java programming language includes various classes that provide an easy-to-use means to access network resources. The classes covered in the **java.net** package are given as follows –

1. **CacheRequest –** The CacheRequest class is used in java whenever there is a need to store resources in ResponseCache. The objects of this class provide an edge for the OutputStream object to store resource data into the cache.

2. **CookieHandler –** The CookieHandler class is used in Java to implement a callback mechanism for securing up an HTTP state management policy implementation inside the HTTP protocol handler. The HTTP state management mechanism specifies the mechanism of how to make HTTP requests and responses.

3. **CookieManager –** The CookieManager class is used to provide a precise implementation of CookieHandler. This class separates the storage of cookies from the policy surrounding accepting and rejecting cookies.

4. **DatagramPacket** – The DatagramPacket class is used to provide a facility for the connectionless transfer of messages from one system to another. This class provides tools for the production of datagram packets for connectionless transmission by applying the datagram socket class.

5. **InetAddress** – The InetAddress class is used to provide methods to get the IP address of any hostname. An IP address is expressed by a 32-bit or 128-bit unsigned number. InetAddress can handle both IPv4 and IPv6 addresses.

6. **Server Socket** – The ServerSocket class is used for implementing system-independent implementation of the server-side of a client/server Socket Connection. The constructor for ServerSocket class throws an exception if it can't listen on the specified port.
7. **Socket** – The Socket class is used to create socket objects that help the users in implementing all fundamental socket operations. The users can implement various networking actions such as sending, reading data, and closing connections.
8. **DatagramSocket** – The DatagramSocket class is a network socket that provides a connection-less point for sending and receiving packets. Every packet sent from a datagram socket is individually routed and delivered. It can further be practiced for transmitting and accepting broadcast information.

9. **Proxy** – A proxy is a changeless object and a kind of tool or method or program or system, which serves to preserve the data of its users and computers. It behaves like a wall between computers and internet users.

10. **URL** – The URL class in Java is the entry point to any available sources on the internet. A Class URL describes a Uniform Resource Locator, which is a signal to a "resource" on the World Wide Web.
11. **URLConnection** – The URLConnection class in Java is an abstract class describing a connection of a resource as defined by a similar URL. The URLConnection class is used for assisting two distinct yet interrelated purposes.

Java Networking Interfaces

The **java.net** package of the Java programming language includes various interfaces also that provide an easy-to-use means to access network resources. The interfaces included in the **java.net** package are as follows:

1. **CookiePolicy** – The CookiePolicy interface in the **java.net** package provides the classes for implementing various networking applications. It decides which cookies should be accepted and which should be rejected

2. **CookieStore –** A CookieStore is an interface that describes a storage space for cookies. CookieManager combines the cookies to the CookieStore for each HTTP response and recovers cookies from the CookieStore for each HTTP request.

3. **FileNameMap –** The FileNameMap interface is an uncomplicated interface that implements a tool to outline a file name and a MIME type string.

4. **SocketOption –** The SocketOption interface helps the users to control the behavior of sockets. Often, it is essential to develop necessary features in Sockets. SocketOptions allows the user to set various standard options.

5. **SocketImplFactory –** The SocketImplFactory interface defines a factory for SocketImpl instances. It is used by the socket class to create socket implementations that implement various policies.

6. **ProtocolFamily –** This interface represents a family of communication protocols. The ProtocolFamily interface contains a method known as name(), which returns the name of the protocol family.

Socket Programming

Java Socket programming is practiced for communication between the applications working on different JRE. Sockets implement the communication tool between two computers using TCP. Java Socket programming can either be connection-oriented or connection-less. In Socket Programming, Socket and ServerSocket classes are managed for connection-oriented socket programming. However, DatagramSocket and DatagramPacket classes are utilized for connection-less socket programming.
A client application generates a socket on its end of the communication and strives to combine that socket with a server. When the connection is established, the server generates an object of socket class on its communication end. The client and the server can now communicate by writing to and reading from the socket.