**Adv. OOP PYQs**

*Authored by Arkajyoti Naskar*

**1. a) Consider s1 and s2 are referring to the instance(s) of same class in Java. Compare: i) s1==s2 and ii) s1. equals(s2) (2)**

**i) s1==s2:** This compares the references of the two objects. It checks if s1 and s2 refer to the exact same object in memory. If they refer to the same instance, the result will be true. If they refer to different instances, even if the contents of the objects are the same, the result will be false.

**ii) s1. equals(s2):** This calls the equals () method defined in the class of the objects being compared. By default, the equals () method in Java compares the references of the objects, similar to the == operator. Therefore, unless the class overrides the equals () method to define custom comparison logic, it will behave the same as s1==s2. If the equals () method is overridden, it can compare the contents of the objects and return true if they are considered equal based on the class's definition.

**b) In java, how can you do certain activity at the time of object destruction? (2)**

In Java, there is no direct equivalent to a destructor as found in languages like C++. However, Java provides a mechanism called finalization, which allows you to perform certain activities when an object is no longer reachable and is eligible for garbage collection. The finalization process involves invoking the finalize () method of an object before it is reclaimed by the garbage collector. The finalize () method allows you to perform clean-up activities at the time of object destruction in Java.

**\*c) In Java, consider a class X has a public method f(int). Class Y extends X and contains a public method f(float). Now consider the following code snippet:**

**int i; float fl;**

**Y c=new Y(); c.f(i); X b=c; b.f(fl);**

**Explain, the calls for method f(). (3)**

An instance of class Y is created and assigned to the variable c. The method f(int) in class Y is invoked using the variable c and passing the integer variable i as an argument. Since the method f(int) is not overridden in class Y, the method in the superclass X is executed.

The error occurs because there is no suitable method in class X that can accept a float argument. The method f(float) is defined in class Y, but when b.f(fl) is called, the reference type is X, so it looks for a matching method in class X. Since there is no f(float) method in class X, the compiler tries to find a compatible method based on the argument type, which is float. However, since f(int) is the closest match (as it can accept an int argument), the compiler tries to perform a conversion from float to int. But this conversion is considered a "possible lossy conversion" because information may be lost in the process.

**d) What is the utility of inner class in Java? (2)**

Inner classes are used to develop a more readable and maintainable code because they logically group classes and interfaces in one place. If there is a class which is a subcomponent of the main class and not used in any other class , only have implementation in that class only, then we can write that subcomponent class as an inner class inside the class where it has implementation and perform some functionality.

**e) What is the use of finally in Java? (1)**

The finally block always executes when the try block exits. This ensures that the finally block is executed even if an unexpected exception occurs. But finally is useful for more than just exception handling — it allows the programmer to avoid having clean-up code accidentally bypassed by a return, continue, or break. Putting clean-up code in a finally block is always a good practice, even when no exceptions are anticipated.

**1) a) What is bytecode in java? (1)**

Bytecode in Java refers to the compiled form of Java source code. When you write Java code, it is compiled by the Java compiler (javac) into a platform-independent binary format called bytecode. Bytecode is a set of instructions that can be executed by the Java Virtual Machine (JVM).

Bytecode is not specific to any particular hardware or operating system. It is designed to be platform-independent, which means that once the Java source code is compiled into bytecode, it can be run on any device or platform that has a Java Virtual Machine.

**\*b) What happens for the following declarations in Java:**

**int x; Sample y; where, Sample is a class. (1) / Consider X is a class, What happens for the statement: X a;**

In Java, when you declare a variable using the syntax int x; you are declaring an integer variable named x. This means that x can hold integer values, such as 1, 10, -5, etc. The variable x will be allocated memory to store the integer value when it is assigned a value.

On the other hand, when you declare a variable using the syntax Sample y;, you are declaring a variable named y of type Sample. In this case, Sample is a class, so y is a reference variable that can refer to objects of the Sample class. It's important to note that declaring a variable of a class type (such as Sample) does not create an object of that class. It only creates a reference variable that can point to objects of that class when they are created.

**c) What is wrapper class in Java? (1)**

A Wrapper class in Java is a class whose object wraps or contains primitive data types. When we create an object to a wrapper class, it contains a field and, in this field, we can store primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

They allow us to modify arguments passed into methods and make code compatible with object-oriented frameworks and APIs that require objects. Wrapper classes also enable the use of primitive types in collection classes, support synchronization in multithreading scenarios, and provide enhanced functionality such as additional methods and the ability to handle null values. Additionally, they facilitate serialization by allowing primitive values to be wrapped and processed as objects. Overall, wrapper classes serve as a bridge between primitive types and objects, enhancing the capabilities and flexibility of Java programs.

**d) Comment on the passing of a primitive data type and an object to a function. (1)**

When passing a primitive data type to a function in Java, the value of the primitive is passed to the function. Any modifications made to the value inside the function will not affect the original value outside the function. This is because primitive data types are passed by value, meaning a copy of the value is created and passed to the function.

On the other hand, when passing an object to a function in Java, the reference to the object is passed to the function. This means that the function can modify the object's state, such as changing its properties or invoking its methods. Any modifications made to the object inside the function will affect the original object outside the function because the reference points to the same object in memory.

In summary, passing a primitive data type to a function creates a copy of the value, while passing an object to a function allows the function to directly manipulate the object's state.

**\*\*b) How the shallow and deep copy of an object can be made in Java? 1.5**

In Java, there are two approaches to copying objects: shallow copy and deep copy.

**Shallow Copy:** When we do a copy of some entity to create two or more than two entities such that changes in one entity are reflected in the other entities as well, then we can say we have done a shallow copy. In shallow copy, new memory allocation never happens for the other entities, and the only reference is copied to the other entities. clone() method of the object class supports shallow copy of the object.

Student s1 = new Student();// creating an object of the class Student

Student s2;

s2 = s1;  // it will copy the reference, not value

**Deep Copy:** A deep copy creates a new object. When we do a copy of some entity to create two or more than two entities such that changes in one entity are not reflected in the other entities, then we can say we have done a deep copy. In the deep copy, a new memory allocation happens for the other entities, and reference is not copied to the other entities. Each entity has its own independent reference.

So for deep copy we need to ensure all the member class also implement the Cloneable interface and override the clone() method of the object class.

Student s1 = new Student();// creating an object of the class Student

Student s2 = new Student(s1);

Public Student(Student s){ name=s.name; roll = s.roll; } //copy constructor.

**\*c) What is the use of final keyword in Java? (1.5)/ Specify the use of final and super in Java. (2.5)**

In Java, the final keyword is used to indicate that a variable, method, or class cannot be modified or overridden, depending on its usage.

When applied to a **variable**, the final keyword indicates that its value cannot be changed once it has been assigned. It makes the variable a constant. When applied to a **method**, the final keyword indicates that the method cannot be overridden by subclasses. When applied to a **class**, the final keyword indicates that the class cannot be subclassed. It prevents other classes from inheriting from the final class,

The **final** keyword provides various benefits in Java, including code robustness, security, and performance optimizations. By using final, you can ensure immutability, enforce method implementation consistency, and restrict class inheritance, depending on the specific needs and design of your code.

The **"super"** keyword in Java is used to refer to the immediate parent class of a subclass. It is often used to access the superclass's members (fields or methods) that have been overridden or hidden by the subclass. By using "super", you can invoke the superclass's constructor, access its methods or fields, and differentiate between superclass and subclass members with the same name.

**\*\*e) What is function overriding in Java and its use? / What is the advantage of using @override? (1)**

Function overriding in Java refers to the ability of a subclass to provide its own implementation of a method that is already defined in its superclass. When a method is overridden, the subclass provides a specific implementation of the method that overrides the implementation in the superclass.

The use of function overriding allows for polymorphism, which means that a subclass object can be treated as an object of its superclass. This provides flexibility and extensibility in the object-oriented paradigm, as it allows different objects of related classes to be used interchangeably while invoking the same method.

The @Override annotation in Java is used to indicate that a method in a subclass is intended to override a method in its superclass. It checks whether a function with same signature exists in the superclass or not and if it is correctly overriding or not.

It is optional but recommended to use @Override to provide explicit documentation and improve code readability.

**g) How can you find the length of array object and String object? (1)**

To find the length of an array object in Java, we can use the length property

int[] numbers = {1, 2, 3, 4, 5}; int length = numbers.length;

To find the length of a String object in Java, we can use the length() method

String message = "Hello, world!"; int length = message.length();

**\*1) a) Java is platform independent -- why? (1.5)**

Java is platform independent because it uses bytecode and the Java Virtual Machine (JVM). Although JVM is not platform independent, it is a virtual machine or a program that provides run-time environment in which java byte code can be executed. The JVM provides a layer of abstraction, handling platform-specific details and making Java programs portable across different operating systems. The use of the same byte code for all JVMs on all platforms make java platform independent. Or java is platform independent because java does not run directly on operating system. It runs on the JVM which you have to install separately.

**\*b) We need to pass the values of an int and a float variable to a method and the method may change the values which are required once we return from the method. How it can be achieved? (1.5)**

In Java, primitive types like int and float are passed by value, which means that the method receives a copy of the original variable's value. If you want the method to modify the values of the variables and have those changes reflected outside the method, you can use wrapper classes or create a custom class to encapsulate the variables. By using a wrapper class or creating a custom class to hold the variables, you can achieve the desired behavior of modifying and retaining the values once the method returns.

**\*f) Compare abstract class and interface in the context of ensuring design guidelines. (1.5) / c) Compare abstract class and interface in Java. 1.5**

Abstract classes can have both abstract and non-abstract methods, while interfaces can only have abstract methods (prior to Java 8). Interfaces can also have default and static methods since Java 8. Abstract classes can have instance variables, constructors, and static methods, whereas interfaces cannot have instance variables (except for static final variables) or constructors. A class can extend only one abstract class, but it can implement multiple interfaces. An abstract class can extend one abstract class and implement multiple Java interfaces. But An interface can extend another Java interface only.

Both abstract classes and interfaces provide a level of abstraction and promote code reusability. Both abstract classes and interfaces can be used to achieve polymorphism and loose coupling. They support modularity and maintainability in the codebase.

**1.\*\*a) How does the task of destructor is accomplished in Java? (1.5)**

In Java, there is no explicit destructor like in languages such as C++. Instead, Java has a garbage collector that automatically manages memory deallocation. In case of Java, periodically a method called garbage collector runs and looks for unreferenced instances and removes those. We can also call the garbage collector explicitly by “System.gc()”. When a garbage collector takes away the object then prior to that finalize() method of corresponding class (if any) is called on behalf of the object.

This automatic memory management eliminates the need for manual destructor implementation in Java.

**\*b) How will you copy the content of one object to another in Java? (1.5)**

To copy the contents of one object to another in Java:

Deep copy creates a new object with separate copies of referenced objects. Deep copy involves recursively copying the object and its references. We can use copy constructor for deep copy.

Student s1 = new Student();// creating an object of the class Student

Student s2 = new Student(s1);//Deep Copy.

Public Student(Student s){ name=s.name; roll = s.roll; } //copy constructor.

**\*c) What will you do to achieve runtime polymorphism in Java? (1.5)**

Method overriding is one of the ways to achieve Runtime Polymorphism in Java. Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.

*Runtime Polymorphism can be achieved through assigning objects created of the derived classes them to variables of the base class type and functions when the base class and derived class functions have same signature. We can then call the overridden methods on these variables, and the appropriate implementation will be invoked based on the actual object type at runtime.* *A superclass reference variable can refer to a subclass object. This is also known as upcasting. Java uses this fact to resolve calls to overridden methods at run time.*

When an overridden method is called through a superclass reference, Java determines which version(superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time.

**\*d) Mention the utility of package in Java. (1.5/2)**

Packages in Java helps in:

1. Grouping logically related components i.e. classes and interfaces.
2. Encapsulation and additional access control.
3. To avoid namespace collision.
4. Reusability and modularity of code and to enhance visibility control.

Packages help in structuring and managing code, ensuring encapsulation, promoting code reuse, and maintaining code integrity.

**\*\*\*\*e) Describe the access specifier for a class in Java. (2 /3)**

The classes and interfaces themselves can have only two access modifiers when declared outside any other class.

**Default:** When we declare a class without any access modifier/ access specifier, it is considered as default or package-private.  because it restricts access to only within the same package.

**Public:** The access level of a public class is everywhere. It can be accessed within the package and outside the package also.

Nested interfaces and classes can have all access modifiers. We cannot declare class/interface with private or protected access modifiers.

**\*\*\*\*f) What is checked exception? (1) /Differentiate checked and unchecked exception. (1.5)/ \*h) How to deal with checked exceptions? (1.5)**

A checked exception in Java is an exception that are checked for handler during compile time. If there is no handler then it will give compilation error so to resolve this we have write a handler or it must be declared in the method signature or handled using a try-catch block. It is a type of exception that the compiler enforces the developer to handle explicitly, either by catching it or declaring it to be thrown.

**Difference between checked and unchecked exceptions:**

***Checked Exceptions:*** Checked exceptions are checked at compile-time by the compiler. They are typically used for exceptional conditions that can be reasonably anticipated and recovered from. Examples include IOException, SQLException, and ClassNotFoundException. Checked exceptions must be either caught using a try-catch block or declared in the method signature using the throws keyword.

***Unchecked Exceptions:*** Unchecked exceptions, also known as runtime exceptions, are not checked at compile-time. They typically represent programming errors or exceptional conditions that are not expected to be handled explicitly.

Examples include NullPointerException, ArrayIndexOutOfBoundsException, and IllegalArgumentException. Unchecked exceptions do not require explicit handling and can be caught or propagated automatically.

**How to deal with checked exceptions:**

Using try-catch block: Surround the code that may throw the checked exception with a try block, followed by one or more catch blocks to handle the exception or perform error recovery.

**public void method() throws ExceptionType {**

**// Code that may throw a checked exception**

**}**

**public void callingMethod() {**

**try {**

**method();**

**} catch (ExceptionType e) {**

**// Exception handling code**

**}**

**}**

Declaring the exception: If the checked exception cannot be handled within the method, it should be declared in the method signature using the throws keyword. This indicates that the method may throw the checked exception, and it becomes the responsibility of the calling method to handle it.

**g) A String object contains only digits. How will you convert into int. (1)**

String numberString = "12345";

int number = Integer.parseInt(numberString);

**b) Justify the non-existence of virtual base class in Java? (1.5)**

Java does not support multiple inheritance of classes. It only allows single inheritance, where a class can extend only one superclass i.e. doesn’t allow any class to have a common ancestor. This design choice eliminates the need for a virtual base class in Java unlike C++.

Instead, Java introduces the concept of interfaces, which allow a class to implement multiple interfaces. Interfaces provide a way to achieve similar functionality as multiple inheritance by specifying a set of methods that a class must implement. However, interfaces do not involve the concept of a virtual base class. Therefore, the non-existence of a virtual base class in Java is justified by its single inheritance model and the use of interfaces to achieve multiple inheritance-like behaviour.

**i) In java, what will you do to check the equality of two objects in a class? (1)**

In Java, to check the equality of two objects in a class, you can override the equals() method provided by the Object class. By default, the equals() method in Java compares the memory references of objects, which may not be the desired behaviour for custom classes. So, to perform a meaningful comparison, we need to override the equals () method in your class and provide our own implementation based on the specific attributes or properties that define equality for your objects.

**\*d) What you can interpret from X$Y.class file? (1)**

The X$Y.class file in Java represents a nested class Y within another class X. When a nested class is compiled, the Java compiler generates a separate class file for the nested class, using the naming convention X$Y.class, where X is the name of the outer class and Y is the name of the nested class.

**Collections & Generics:**

**b) Suppose a collection in Java holds STUDENT objects. In order to use the contains() method of the collection , explain the measures that will you take? (3)**

In order to use the contains() method of a collection in Java to check for the presence of STUDENT objects, you need to ensure that the equals() and hashCode() methods are properly implemented in the STUDENT class. Override the equals() method to compare the relevant attributes of a STUDENT object for equality. This method should return true if the attributes of two STUDENT objects are equal, and false otherwise.

Override the hashCode() method to generate a unique hash code for each STUDENT object. This method is used by collections such as HashSet and HashMap to efficiently store and retrieve objects. By properly implementing the equals() and hashCode() methods, you ensure that the contains() method accurately identifies whether a STUDENT object is present in the collection.

**7. a) How does interface help in designing a solution? How does concept of package help in organizing a software solution? (3)**

Interface plays a crucial role in designing a solution by providing a contract or blueprint for classes to follow. It defines a set of methods that must be implemented by any class that implements the interface. This promotes code consistency, modularity, and allows for easy substitution of different implementations. By using interfaces, you can abstract away the specific implementation details and focus on the desired functionality. Interfaces also facilitate loose coupling, as classes can interact with each other through interfaces rather than concrete implementations

.

The concept of a package in Java helps in organizing a software solution by providing a way to group related classes, interfaces, and resources together. Packages act as containers for classes, allowing you to categorize and structure your codebase. This promotes code organization, reusability, and maintainability. Packages provide a hierarchical structure, allowing for better organization of code into modules, subsystems, or components. Additionally, packages enable access control through their visibility modifiers (e.g., public, private, protected), helping to encapsulate and control the accessibility of classes and their members. By using packages, you can manage dependencies, reduce naming conflicts, and make your codebase more manageable and comprehensible.

**b) Suppose a collection in Java holds STUDENT objects. STUDENT has roll, name and score as attributes. Depending on the requirement collection may be sorted on roll (ascending/descending) or score (ascending/descending). Write down the code snippet to support the requirement. (5)**

**import java.util.\*;**

**class Student {**

**private int roll, score;**

**private String name;**

**// Constructor and getters/setters**

**// Other methods}**

**public class StudentSortingExample {**

**public static void main(String[] args) {**

**// Create a list of STUDENT objects**

**List<Student> studentList = new ArrayList<>();**

**// Add STUDENT objects to the list**

**// Sort the list based on roll in ascending order**

**Collections.sort(studentList, new Comparator<Student>() {**

**public int compare(Student s1, Student s2) {**

**return s1.getRoll() - s2.getRoll();**

**}**

**});**

**// Sort the list based on roll in descending order**

**Collections.sort(studentList, new Comparator<Student>() {**

**public int compare(Student s1, Student s2) {**

**return s2.getRoll() - s1.getRoll();**

**}**

**});**

**// Sort the list based on score in ascending order**

**Collections.sort(studentList, new Comparator<Student>() {**

**public int compare(Student s1, Student s2) {**

**return s1.getScore() - s2.getScore();**

**}**

**});**

**// Sort the list based on score in descending order**

**Collections.sort(studentList, new Comparator<Student>() {**

**public int compare(Student s1, Student s2) {**

**return s2.getScore() - s1.getScore();**

**}**

**});**

**}**

**}**

**c) How can you set the priority for the elements in a priority queue? (2)**

By default, a priority queue uses the natural ordering of elements, which means the elements should implement the Comparable interface. The Comparable interface defines a compareTo() method that determines the natural ordering of objects. The priority queue will automatically prioritize elements based on their natural ordering.

If you want to define a custom ordering for the elements in the priority queue, we can create a separate class that implements the Comparator interface. The Comparator interface has a compare() method that allows us to define custom comparison logic. We can then create a priority queue instance by passing the custom comparator as an argument. The priority queue will use the custom comparator to prioritize elements according to our defined logic.

In summary, we can set the priority for elements in a priority queue by either implementing the Comparable interface in the element class or by providing a custom Comparator implementation when creating the priority queue.

**\*4) a) Write a code snippet in Java to find out the number of words in a string. Assume words are separated by one space. [C-like solution will not be considered]. (2.5)**

**public class WordCountExample {**

**public static void main(String[] args) {**

**String inputString = "This is a sample string";**

**int wordCount = countWords(inputString);**

**System.out.println("Number of words: " + wordCount);**

**}**

**public static int countWords(String str) {**

**if (str == null || str.isEmpty()) {**

**return 0;**

**}**

**// Split the string into words using space as the delimiter**

**String[] words = str.split(" ");**

**// Return the count of words**

**return words.length;**

**}**

**}**

**find the longest word, whether “abcd” is present in the input text or not and display all the words also. Write the code.(4)**

**public class WordManipulationExample {**

**public static void main(String[] args) {**

**String inputString = "This is a sample text multiple words";**

**// Find the longest word**

**String longestWord = findLongestWord(inputString);**

**System.out.println("Longest word: " + longestWord);**

**// Check if "abcd" is present**

**boolean containsAbcd = containsWord(inputString, "abcd");**

**System.out.println("Contains 'abcd': " + containsAbcd);**

**// Display all words**

**String[] words = getWords(inputString);**

**System.out.println("All words:");**

**for (String word : words)**

**System.out.println(word);**

**}**

**public static String findLongestWord(String str) {**

**String[] words = getWords(str);**

**String longestWord = "";**

**for (String word : words)**

**if (word.length() > longestWord.length()) longestWord = word;**

**return longestWord;**

**}**

**public static boolean containsWord(String str, String word) {**

**String[] words = getWords(str);**

**for (String w : words)**

**if (w.equals(word)) return true;**

**return false;**

**}**

**public static String[] getWords(String str) {**

**return str.split(" ");**

**}**

**}**

**b) Roll and score of the students are to be stored in a suitable data structure where frequently one may like to know the score against a roll. Suggest the scheme in Java with justification.** (1)

One suitable data structure to store the roll and score of students, where the score can be easily retrieved based on the roll, is a HashMap<Integer, Integer> in Java.

HashMap<Integer, Integer> studentScores = new HashMap<>();

HashMap provides efficient retrieval of values based on keys, allowing constant-time access to scores based on roll numbers. HashMap allows easy association between roll numbers and scores, with each roll number serving as a unique identifier. It allows for dynamic resizing and can accommodate a varying number of students. It can easily handle additions, deletions, and updates to student records. HashMap optimizes memory usage by storing only necessary.

**\*\*4) a) Roll and score of the students are to be stored in a suitable data structure where frequently one may like to know the score against a roll. Suggest the scheme in Java with justification and show how to store and retrieve score for a query. (3)**

**import java.util.HashMap;**

**public class StudentScores {**

**private HashMap<Integer, Integer> scores;**

**public StudentScores() {**

**scores = new HashMap<>();**

**}**

**public void addScore(int rollNumber, int score) {**

**scores.put(rollNumber, score);**

**}**

**public int getScore(int rollNumber) {**

**return scores.getOrDefault(rollNumber, -1);**

**}**

**// Other methods...**

**public static void main(String[] args) {**

**StudentScores studentScores = new StudentScores();**

**// Storing scores**

**studentScores.addScore(101, 85);**

**studentScores.addScore(102, 92);**

**studentScores.addScore(103, 78);**

**// Retrieving scores**

**int score101 = studentScores.getScore(101);**

**System.out.println("Score for roll number 101: " + score101);**

**int score102 = studentScores.getScore(102);**

**System.out.println("Score for roll number 102: " + score102);**

**int score105 = studentScores.getScore(105);**

**System.out.println("Score for roll number 105: " + score105);**

**// Output: -1 (not found)**

**}**

**}**

**b) What is the difference between array and ArrayList? (1)**

Arrays have a fixed size and contains homogeneous data, while ArrayLists can dynamically grow or shrink in size. Arrays offer direct element access based on index, while ArrayLists provide convenient methods for element manipulation and can also store elements of different type.

**\*b) For every friend of yours name and date of birth (taken as string) are stored in an array. Design the class(es) and write down the code for the method to find the name of the friends born in a particular month (given as int argument to the method). Date string may follow any of the following formats: dd/mm/yyyy or dd-mm-yyyy format.(4)**

**import java.util.ArrayList;**

**import java.util.List;**

**class Friend {**

**private String name;**

**private String dateOfBirth;**

**public Friend(String name, String dateOfBirth) {**

**this.name = name;**

**this.dateOfBirth = dateOfBirth;**

**}**

**public String getName() {**

**return name;**

**}**

**public String getDateOfBirth() {**

**return dateOfBirth;**

**}**

**}**

**class FriendManager {**

**private List<Friend> friends;**

**public FriendManager() {**

**friends = new ArrayList<>();**

**}**

**public void addFriend(Friend friend) {**

**friends.add(friend);**

**}**

**public List<String> getFriendsByMonth(int month) {**

**List<String> friendsInMonth = new ArrayList<>();**

**for (Friend friend : friends) {**

**String dateOfBirth = friend.getDateOfBirth();**

**String[] parts = dateOfBirth.split("[/-]");**

**int friendMonth = Integer.parseInt(parts[1]);**

**if (friendMonth == month)**

**friendsInMonth.add(friend.getName());**

**}**

**return friendsInMonth;**

**}**

**}**

**public class Main {**

**public static void main(String[] args) {**

**FriendManager friendManager = new FriendManager();**

**// Add friends to the friend manager**

**friendManager.addFriend(new Friend("John", "12/05/1990"));**

**friendManager.addFriend(new Friend("Emily", "03-08-1995"));**

**friendManager.addFriend(new Friend("David", "20-05-1992"));**

**friendManager.addFriend(new Friend("Sarah", "10/10/1994"));**

**// Get friends born in a specific month**

**List<String> friendsInMay =friendManager.getFriendsByMonth(5);**

**System.out.println("Friends born in May:");**

**for (String friend : friendsInMay) {**

**System.out.println(friend);**

**}**

**}**

**}**

**Threading:**

**\*2. a) How can you specify the code for a thread and data on which it works? (3) / How can you create the threads using Runnable? Clearly indicate the code and the data on which the thread will work. (2.5)**

In Java, you can specify the code for a thread and the data it works on by implementing the Runnable interface.

**public class MyThread implements Runnable {**

**private int threadId;**

**private Data data;**

**public MyThread(int threadId, Data data) {**

**this.threadId = threadId;**

**this.data = data;**

**}**

**@Override**

**public void run() {**

**// Code to be executed by the thread**

**System.out.println("Thread " + threadId + " is working on data: " + data);**

**// Perform operations on the data**

**}**

**}**

In the above example, Data represents the data on which each thread will work. The run() method contains the code that will be executed when the thread is started. It prints the thread ID and the data it's working on.

**public class Main {**

**public static void main(String[] args) {**

**Data data1 = new Data("Data 1");**

**Data data2 = new Data("Data 2");**

**MyThread thread1 = new MyThread(1, data1);**

**MyThread thread2 = new MyThread(2, data2);**

**Thread t1 = new Thread(thread1);**

**Thread t2 = new Thread(thread2);**

**t1.start(); // Start thread 1**

**t2.start(); // Start thread 2**

**}**

**}**

In the main method, we create two instances of Data with different values. Then, we create two instances of MyThread, passing the thread ID and the respective data object. Next, we create Thread instances t1 and t2, passing the corresponding MyThread instances. Finally, we call start() on each thread to begin their execution.

**b) notifyAll() may be preferred over notify() – explain with a scenario. (4)**

In Java, the notify() and notifyAll() methods are used in conjunction with the wait() method to implement inter-thread communication within a synchronized block or method. if multiple threads are waiting on any locks in Java, notify method sends a notification to only one of the waiting threads while notifyAll informs all threads waiting on that lock. In a way, the notifyAll method is safer because it sends a notification to all threads, so if any thread misses the notification, there are other threads to do the job, while in the case of notify() method if the notified thread misses the notification then it could create subtle, hard to debug issues.

**public class Example {**

**private boolean condition = false;**

**public synchronized void waitForCondition() throws InterruptedException {**

**while (!condition) {**

**wait(); // Wait until the condition is satisfied**

**}**

**System.out.println("Condition satisfied!");**

**}**

**public synchronized void setCondition() {**

**condition = true;**

**notifyAll(); // Notify all waiting threads**

**}**

**}**

**\*2) a) Compare start() and run(). A runnable object makes a call to run method -- What will happen? (2)**

In Java, the start() and run() methods are related to multithreading.

**start():** It is a method defined in the Thread class and is used to start a new thread of execution. When start() is called on a Thread object, it creates a new thread and invokes the run() method of that thread in a separate thread of execution. This allows concurrent execution of multiple threads.

**run():** It is a method defined in the Runnable interface, which is implemented by a class that represents a task to be executed by a thread. The run() method contains the code that defines the task's behavior. However, if you directly call the run() method on a Runnable object, it does not start a new thread; instead, it executes the code synchronously within the current thread.

If a Runnable object makes a direct call to the run() method, without invoking start() on a Thread object, the run() method will be executed synchronously in the current thread. This means that the code inside the run() method will be executed sequentially, without creating a new thread. This behavior is similar to a regular method call and does not provide the benefits of concurrent execution.

**b) Suppose there is a list of accounts. Each account has account number and balance. Multiple threads are there to work on the same account list and will update the account balance. How will you achieve proper update allowing the concurrency to the extent possible? Provide the design. (4)**

To achieve proper update and allow concurrency to the extent possible in a scenario where multiple threads are updating the account balances in a shared account list, you can use synchronization mechanisms in Java. One way to achieve this is by using the synchronized keyword along with a lock mechanism.

**public class Account {**

**private int accountNumber;**

**private double balance;**

**public Account(int accountNumber, double balance) {**

**this.accountNumber = accountNumber; this.balance = balance;**

**}**

**public synchronized void updateBalance(double amount) {**

**balance += amount;**

**}**

**// Other methods and getters/setters**

**}**

**import java.util.ArrayList;**

**import java.util.List;**

**public class Bank {**

**private List<Account> accountList;**

**public Bank() {**

**accountList = new ArrayList<>();**

**}**

**public void updateAccountBalance(int accountNumber, double amount) {**

**Account account = getAccountByNumber(accountNumber);**

**if (account != null) {**

**synchronized (account) {**

**account.updateBalance(amount);**

**}**

**}**

**}**

**private Account getAccountByNumber(int accountNumber) {**

**// Find and return the account from the list**

**}**

**// Other methods and functionalities**

**}**

By using synchronization with the synchronized keyword and appropriate locking mechanisms, the updateAccountBalance() method will be executed by multiple threads in a synchronized manner. This ensures that only one thread can update an account balance at a time, avoiding conflicts and ensuring proper updates to the account balances.

**public class Main {**

**public static void main(String[] args) throws InterruptedException {**

**Bank bank = new Bank();**

**// Create multiple threads to update account balances**

**Thread thread1 = new Thread(() -> bank.updateAccountBalance(123, 100));**

**Thread thread2 = new Thread(() -> bank.updateAccountBalance(456, -50));**

**// Start the threads**

**thread1.start();thread2.start();**

**// Wait for the threads to complete**

**thread1.join();thread2.join();**

**// Perform other operations or display account balances**

**}**

**}**

1. **\*a) What are two approaches for creating threads. Which one is preferred and why? (2)**

In Java, there are two approaches for creating threads:

1. ***Extending the Thread class:*** This approach involves creating a new class that extends the Thread class and overrides the run method. Instances of this class can then be created and started as separate threads.
2. ***Implementing the Runnable interface:*** This approach involves creating a class that implements the Runnable interface and overrides the run() method. Instances of this class can be passed as parameters to Thread objects and started as separate threads.

The preferred approach is implementing the Runnable interface because it offers better flexibility and code reusability. By implementing Runnable, you can separate the thread's behavior from the actual class hierarchy, allowing multiple classes to share the same runnable object and promoting a more modular and maintainable code structure. Additionally, implementing Runnable is preferred over extending Thread because Java supports single inheritance, and by implementing Runnable, you can still inherit from other classes if needed.

**\*b) Suppose there is a predesigned class Data that has a method modify() to change the value of attribute. Now, in a multithreaded environment number of threads with same Data object and may call modify(). What measures will you take to prevent simultaneous attempt of modification of Data object. Describe with skeleton code. (2.5/3)**

To prevent simultaneous modification of a shared Data object in a multithreaded environment, we can use synchronization mechanisms such as the synchronized statement if there is no synchronized methods.

**class Data {**

**private int attribute;**

**public void modify() {**

**// Code to modify the attribute**

**}**

**}**

**public class Info implements runnable{**

**Data d;**

**Info(Data a){ d=a; }**

**public void run(){**

**synchronized(d){**

**d.modify();**

**}**

**}**

**}**

This prevents simultaneous attempts to modify the object and ensures thread-safety.

**3. There is a list of items that stores item code, item name, rate and quantity stock for different items. Multiple viewers can view the information without any restriction. But update (changing the quantity stock) on a specific item cannot be done simultaneously. Simultaneous update on different items can take place. Design the classes and write down the skeleton code. (7) /A stocklist contains item code and quantity available for different items. Salespersons update the stock. Updating the stock for different items may be simultaneous. But, Care is to be taken to resist the simultaneous update of same item. Design the necessary classes. Skeleton of the code/textual description may be provided to make your design understandable.(5)**

**public class Item {**

**private String itemCode;**

**private String itemName;**

**private double rate;**

**private int quantityStock;**

**public Item(String itemCode, String itemName, double rate, int quantityStock) {**

**this.itemCode = itemCode; this.itemName = itemName;**

**this.rate = rate; this.quantityStock = quantityStock;**

**}**

**public String getItemCode() {**

**return itemCode;**

**}**

**public String getItemName() {**

**return itemName;**

**}**

**public double getRate() {**

**return rate;**

**}**

**public synchronized int getQuantityStock() {**

**return quantityStock;**

**}**

**public synchronized void setQuantityStock(int quantityStock) {**

**this.quantityStock = quantityStock;**

**}**

**}**

**class ItemList {**

**private List<Item> items;**

**public ItemList() {**

**this.items = new ArrayList<>();**

**}**

**public synchronized void addItem(Item item) {**

**items.add(item);**

**}**

**public synchronized Item getItemByCode(String itemCode) {**

**for (Item item : items) {**

**if (item.getItemCode().equals(itemCode))**

**return item;**

**}**

**return null; // Item not found**

**}**

**public synchronized void updateItemStock(String itemCode, int newQuantity) {**

**Item item = getItemByCode(itemCode);**

**if (item != null) {**

**item.setQuantityStock(newQuantity);**

**System.out.println("Quantity stock updated for item " + itemCode);**

**} else System.out.println("Item not found.");**

**}**

**}**

**class Viewer extends Thread {**

**private ItemList itemList;**

**private String itemCode;**

**public Viewer(ItemList itemList, String itemCode) {**

**this.itemList = itemList;this.itemCode = itemCode;**

**}**

**public void run() {**

**Item item = itemList.getItemByCode(itemCode);**

**if (item != null) {**

**System.out.println("Viewer: Item Code: " + item.getItemCode());**

**System.out.println("Viewer: Item Name: " + item.getItemName());**

**System.out.println("Viewer: Rate: " + item.getRate());**

**System.out.println("Viewer: Quantity Stock:"+ item.getQuantityStock());**

**} else System.out.println("Viewer: Item not found.");**

**}**

**}**

**class UpdateThread extends Thread {**

**private ItemList itemList; private String itemCode;**

**private int newQuantity;**

**public UpdateThread(ItemList itemList, String itemCode, int newQuantity) {**

**this.itemList = itemList;this.itemCode = itemCode;**

**this.newQuantity = newQuantity;**

**}**

**public void run() {**

**itemList.updateItemStock(itemCode, newQuantity);**

**}**

**}**

**public class Main {**

**public static void main(String[] args) {**

**ItemList itemList = new ItemList();**

**// Add items to the list**

**Item item1 = new Item("001", "Item 1", 10.0, 50);**

**Item item2 = new Item("002", "Item 2", 15.0, 75);**

**Item item3 = new Item("003", "Item 3", 20.0, 100);**

**itemList.addItem(item1);itemList.addItem(item2);**

**itemList.addItem(item3);**

**// Create viewer threads**

**Viewer viewer1 = new Viewer(itemList, "001");**

**Viewer viewer2 = new Viewer(itemList, "002");**

**Viewer viewer3 = new Viewer(itemList, "003");**

**// Start viewer threads**

**viewer1.start(); viewer2.start();viewer3.start();**

**// Create update threads**

**UpdateThread updateThread1 = new UpdateThread(itemList, "001", 25);**

**UpdateThread updateThread2 = new UpdateThread(itemList, "002", 50);**

**// Start update threads**

**updateThread1.start(); updateThread2.start();**

**}**

**}**

**#2**

**import java.util.ArrayList;**

**import java.util.List;**

**class Item {**

**private String itemCode;**

**private int quantity;**

**public Item(String itemCode, int quantity) {**

**this.itemCode = itemCode;**

**this.quantity = quantity;**

**}**

**public synchronized void updateQuantity(int quantity) {**

**this.quantity += quantity;**

**System.out.println("Stock updated for item code: " + itemCode);**

**}**

**public synchronized int getQuantity() {**

**return quantity;**

**}**

**public String getItemCode() {**

**return itemCode;**

**}**

**}**

**class StockList {**

**private List<Item> items;**

**public StockList() {**

**items = new ArrayList<>();**

**}**

**public void addItem(Item item) {**

**items.add(item);**

**}**

**public synchronized void updateStock(String itemCode, int quantity) {**

**for (Item item : items) {**

**if (item.getItemCode().equals(itemCode)) {**

**item.updateQuantity(quantity);**

**return;**

**}**

**}**

**System.out.println("Item not found in stock: " + itemCode);**

**}**

**public synchronized int getStock(String itemCode) {**

**for (Item item : items) {**

**if (item.getItemCode().equals(itemCode))**

**return item.getQuantity();**

**}**

**System.out.println("Item not found in stock: " + itemCode);**

**return 0;**

**}**

**}**

**class Salesperson extends Thread {**

**private StockList stockList;**

**private String itemCode;**

**private int quantity;**

**public Salesperson(StockList stockList, String itemCode, int quantity) {**

**this.stockList = stockList; this.itemCode = itemCode;**

**this.quantity = quantity;**

**}**

**public void run() {**

**stockList.updateStock(itemCode, quantity);**

**}**

**}**

**public class Main {**

**public static void main(String[] args) {**

**StockList stockList = new StockList();**

**// Create initial items and add them to the stock list**

**Item item1 = new Item("001", 10);Item item2 = new Item("002", 5);**

**stockList.addItem(item1);stockList.addItem(item2);**

**// Create salesperson threads to update the stock**

**Salesperson salesperson1 = new Salesperson(stockList, "001", 3);**

**Salesperson salesperson2 = new Salesperson(stockList, "002", -2);**

**Salesperson salesperson3 = new Salesperson(stockList, "003", 7);**

**// Start salesperson threads**

**salesperson1.start();salesperson2.start();salesperson3.start();**

**// Wait for all salesperson threads to complete**

**try {**

**salesperson1.join(); salesperson2.join(); salesperson3.join();**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**// Print the final stock**

**System.out.println("Final Stock:");**

**System.out.println("Item 001: " + stockList.getStock("001"));**

**System.out.println("Item 002: " + stockList.getStock("002"));**

**System.out.println("Item 003: " + stockList.getStock("003"));**

**}**

**}**

**CLASS DESIGN:**

**\*d) Anybody designing a course of a curriculum must follow certain specifications like, predefined maximum and minimum contact hours for the course. One must provide the content of the course, lecture plan and text books. What measures you will take to enforce all these? (1.5/2)**

**\*3) a) A software system deals with basic objects of types like, Employee, Dept, Product, StockInfo, SalesInfo, PurchaseInfo. The system has different functional modules like, EmployeeManagement, SalesManagement, PurchaseManagement. Modules work with the subsets of the basic objects. How will you organize your system? (2.5 /3)**

**FILE & I/O:**

**3. Consider each student object has roll, name and phone number. One should be able to collect data for the students and display the same. The objects are also to be stored in a file and all are to be read. Design the class(es) and write code to work with the file. (8)**

**\*c) Accept a filename from user. If it is a directory then display the files in it else display a message “not a directory”. 3 / Accept a filename from user and check whether it’s a directory or not. (1.5)**

**\*6. a) Suppose there is a user defined class STUDENT with roll, name and score as attributes. Objects are to be stored in a binary file. Design the necessary class for that. Write down the code in Java to display all the objects from the file. (7)**

**GUI:**

**4.** **a) Compare panel and frame. (3)**

**b) Write down the code snippet for an applet/application to display a window for the following. A set of options are to be shown and user can select only one from those options. Once user clicks on OK button, the selected option has to be displayed on a message box. (5)**

or

**5. a) In a GUI program in java how can you make a text field read only? In such case how can you put the content there? (2)**

**b) Consider a list kind of interface in GUI program in Java that allows multi selection. Write down the code snippet for an applet/application to add a scrollbar with it and also to find out the selected items. (6 )**

**5) a) What is a frame in Java? (1.5)**

**b) Write down the steps for event handling.**