1. What is threads, and how to create a thread?

Answer: Java threads are like separate lines of execution in a program that allows it to do a multiple things at the same time. They're used to perform tasks simultaneously

Java threads are like having multiple hands working on different tasks within your computer

program at the same time.

For Example: Imagine you're cooking in the kitchen. You have multiple pots on the stove,

and each pot is like a thread. You can mix one pot while another pot is simmering, making

your cooking process more efficient.

public class Main {

public static void main(String[] args) {

// Thread 1: Frying eggs

Thread eggThread = new Thread(new Runnable() {

public void run() {

for (int i = 1; i <= 5; i++) {

System.out.println("Frying egg " + i);

}

}

});

// Thread 2: Toasting bread

Thread toastThread = new Thread(new Runnable() {

public void run() {

for (int i = 1; i <= 5; i++) {

System.out.println("Toasting bread slice " + i);

}

}

});

// Thread 3: Brewing coffee

Thread coffeeThread = new Thread(new Runnable() {

public void run() {

for (int i = 1; i <= 5; i++) {

System.out.println("Brewing coffee cup " + i);

}

}

});

// Start all threads to make breakfast simultaneously

eggThread.start();

toastThread.start();

coffeeThread.start();

}

}

==> There are two ways to create a thread.

\*\* It can be created by 'extending' the Thread class and overriding its run() method:

---- Extend Syntax ... This is the syntax for extending the thread

\*\* Another way to create a thread is to 'implement' the Runnable interface:

---- Implement Syntax ... This is the syntax for Implementing the thread

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

2. What is Lambda expressions and it's Benefits ?

Answer: Lambda expressions are a feature introduced in Java 8.

It gives a brief(concise) way to represent one-method interfaces that is functional interfaces.

They enable the use of functional programming concepts in Java.

They're great for doing quick, specific tasks.

They make your code easier to read.

==> Syntax of Lambda Expressions

A lambda expression has three parts:

Parameters: A set of parameters inside the parentheses.

Arrow token: " ->"

Body: An expression or a block of code witten in curly braces.

==> Functional Interfaces

Lambda expressions work with functional interfaces.

A functional interface is an interface with exactly one abstract method.

Examples: Runnable, Callable, ActionListener.

==> Benefits of Lambda Expressions

Conciseness: Reduced boilerplate code.

Readability: More expressive and natural.

Flexibility: Easier to work with functional interfaces.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

3. What is collections?

Answer: In Java, the term "Collections" typically refers to the Java Collections Framework,

which is a set of classes and interfaces in the java.util package.

Key components of the Java Collections Framework include:

\*\* Interfaces:

\* Collection:

The root interface for all collection types, representing a group of objects.

\* List:

An ordered collection where elements can be accessed by index. Implementations

include ArrayList, LinkedList, and Vector.

\* Set:

A collection that does not allow duplicate elements. Implementations include HashSet,

LinkedHashSet, and TreeSet.

\* Map:

An object that maps keys to values. Implementations include HashMap, LinkedHashMap, and TreeMap.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

4. What is Function overriding and overloading?

Answer:

Function Overloading:

\* Function overloading is when multiple functions in the same scope have the same name

but different parameters.

Function Overriding:

\* Function overriding occurs in inheritance when a subclass provides a specific

implementation for a method that is already defined in its superclass.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

5. What is Method Overloading ?

Answer: In Java, method overloading occurs when a class has multiple methods with

the same name but different parameters (number, type, or order).

Example:

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public double add(double a, double b) {

return a + b;

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

6. What is Method Overriding ?

Answer: In Java, method overriding happens when a subclass provides a specific

implementation for a method that is already defined in its superclass.

When you have a class (let's call it a 'superclass') with a certain action or behavior, and

then you create another class (a 'subclass') that wants to do the same action but in its

own way, that's called method overriding in Java.

Example:

class Animal {

public void makeSound() {

System.out.println("Some generic sound");

}

}

class Dog extends Animal {

@Override

public void makeSound() {

System.out.println("Bark! Bark!");

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

7. What is Garbage collection?

Answer: Garbage collection is an automatic memory management process in

programming languages where the system identifies and frees up memory

occupied by objects that are no longer in use. This helps prevent memory leaks

and allows developers to focus on writing code without explicitly deallocating memory.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

8. What is Access modifiers?

Answer: Access modifiers are keywords that can be used to control the visibility of fields,

methods, and constructors in a class. The four access modifiers in Java are

public, protected, default, and private.

\* Private: The access level of a private modifier is only within the class.

It cannot be accessed from outside the class.

\* Protected: The access level of a protected modifier is within the package and outside

the package through child class. If you do not make the child class, it cannot be

accessed from outside the package.

\* Default: The access level of a default modifier is only within the package. It cannot be

accessed from outside the package. If you do not specify any access level, it will be the default.

\* Public: The access level of a public modifier is everywhere. It can be accessed from within the class,

outside the class, within the package and outside the package.

Private : Accessible from the class where they are defined.

Protected : Accessible from the classes, package, subclass and from within the class.

Default : Accessible from all the class.

Public : The data items and functions are accessible from anywhere.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

9. What is package?

Answer: In Java, a package is a way to organize related classes and interfaces into a single directory hierarchy.

It helps in structuring and managing the code in a modular and organized manner. Packages provide a

means to group related components and avoid naming conflicts.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

10. What is Recursion?

Answer: Recursion in Java is a programming technique where a method calls itself to solve a

smaller instance of the same problem. In simpler terms, it's like a function looking

into a mirror and finding a smaller version of itself, repeating the process until the

problem becomes small enough to solve directly. Recursion often involves breaking

down a complex problem into simpler, similar sub-problems and solving them incrementally.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

11. What is aggregation?

Answer: Aggregation in object-oriented programming is a "has-a" relationship where one class

contains another class as a part, allowing the contained object to exist independently and be

shared among multiple containers. It represents a weaker association compared to composition.

==> Aggregation is a type of association in object-oriented programming that represents a

"whole-part" relationship between two classes. In aggregation, one class contains an object of

another class, and there is a weaker relationship between them compared to composition.

The contained object can exist independently of the container, and it can be shared among

multiple containers.

\*\* The main characteristics of aggregation are:

\* Existence Independence:

--> The "part" (object) can exist independently of the "whole" (container). If the container is destroyed,

the part can still exist.

\* Multiplicity:

--> It allows one-to-one, one-to-many, or many-to-many relationships between the container and

the contained object.

\* Flexibility:

--> The contained object can be shared among multiple containers, providing more flexibility than

composition.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

12. What is composition?

Answer: Composition is a strong form of association in object-oriented programming, where one class

contains another class as a part, and the contained object has a lifecycle dependent on the container.

In composition, the relationship is a "whole-part" connection, and if the container is destroyed,

the contained object is also destroyed.

\*\* Key characteristics of composition include:

\* Existence Dependency:

--> The contained object cannot exist independently of the container. When the container is

created or destroyed, it directly affects the contained object.

\* Strong Association:

The relationship between the container and the contained object is considered strong, and

typically the contained object is specifically designed to be used only by the container.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

13. What is the difference between this & super keyword?

Answer: The "this" keyword in Java refers to the current instance of a class and is used to differentiate

between instance variables and local variables. The "super" keyword is used to invoke the superclass

methods or access the superclass fields from a subclass.

==> this keyword:

\* Refers to the current instance of a class.

\* Used to differentiate between instance variables and local variables, and to invoke the current

class methods.

==> super keyword:

\* Refers to the superclass of a class.

\* Used to invoke the superclass methods, access the superclass fields, and call the superclass

constructor from a subclass.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

14. What are the main use of "this" keyword?

Answer: The main uses of the "this" keyword in Java are to:

1. Refer to the Current Instance:

==> It allows differentiation between instance variables and local variables when they share the

same name within a method or constructor.

2. Invoke Current Class Methods:

==> It is used to call other methods or constructors within the same class, particularly when there

is a need to disambiguate between instance variables and parameters with the same name.

3. Pass the Current Object:

==> It enables passing the current object as a parameter to other methods, facilitating operations

on the object within those methods.