Basic Java Concepts

Q1: What are the key features of Java?

Platform Independent – Java programs run on any OS with JVM (Write Once, Run Anywhere).

Object-Oriented – Uses concepts like encapsulation, inheritance, and polymorphism.

Secure & Robust – No direct memory access, automatic garbage collection, and exception handling.

Multithreading Support – Multiple tasks can run concurrently.

High Performance – JIT (Just-In-Time) compiler improves execution speed.

Automatic Memory Management – Garbage Collection removes unused objects.

Q2: What is the difference between JDK, JRE, and JVM?

Component	Description
JDK (Java Development Kit)	Contains compiler (`javac`), debugger, and JRE.
JRE (Java Runtime Environment)	Includes JVM + libraries to run Java applications.
JVM (Java Virtual Machine)	Converts bytecode into machine code for execution.

Example:

If you **only want to run** a Java program → You need **JRE**.

If you **want to develop & run** Java programs → You need **JDK**.

Object Oriented Programming

Q3: What are the four pillars of OOP?

Encapsulation – Wrapping data & methods together.

Inheritance – One class acquiring properties of another.

Polymorphism – A single interface, multiple implementations.

Abstraction – Hiding implementation details from the user.

Q4: What is Encapsulation?

Encapsulation ensures data security by using **private** fields and providing **public** getters & setters.

```
class Student {
private String name;  // Data Hiding

public String getName() { return name; }
public void setName(String name) { this.name = name; }
}

public class Main {
public static void main(String[] args) {
      Student s = new Student();
      s.setName("John");
      System.out.println(s.getName());  // Output: John
}
}
```

Q5: What is Inheritance?

Inheritance allows a class to **inherit** attributes and methods from another class.

```
class Animal {
void makeSound() { System.out.println("Animal makes a sound"); }
}

class Dog extends Animal {  // Dog inherits Animal
void bark() { System.out.println("Dog barks"); }
}

public class Main {
public static void main(String[] args) {
      Dog d = new Dog();
      d.makeSound();  // Inherited method
      d.bark();       // Dog-specific method
}
}
```

Q6: What is the difference between Overloading and Overriding?

Feature	Method Overloading	Method Overriding
Definition	Same method name, different parameters	Same method name, same parameters
Where?	Same class	Parent and child classes
Return Type?	Can be different	Must be same or covariant
Access Modifier?	No restriction	Cannot reduce visibility

Overloading Example:

```
class MathUtils {
int add(int a, int b) { return a + b; }
```

```
double add(double a, double b) { return a + b; }
}
```

Overriding Example:

```
class Parent {
void show() { System.out.println("Parent class"); }
}

class Child extends Parent {
void show() { System.out.println("Child class"); } // Overriding
}
```

Q7: Abstract Class vs. Interface

Feature	Abstract Class	Interface
Methods	Can have both abstract & concrete methods	Only abstract methods (until Java 8)
Variables	Can have instance variables	Only constants (static & final)
Access Modifiers	Supports public, private, protected	Only public methods
Multiple Inheritance?	No	Yes (A class can implement multiple interfaces)

Example:

```
abstract class Animal {
abstract void sound();  // Abstract method
void eat() { System.out.println("Eating..."); }  // Concrete method
}

class Dog extends Animal {
void sound() { System.out.println("Dog barks"); } // Implementing
abstract method
}
```

Memory Management & Exception Handling

Q8: What is Garbage Collection (GC)?

Java automatically removes unused objects using **Garbage Collection (GC)**.

Ways to request GC:

```
System.gc();  // Suggests JVM to run GC
```

Q9: Checked vs. Unchecked Exceptions

Type	Checked Exception	Unchecked Exception
When?	At compile-time	At runtime

Type	Checked Exception	Unchecked Exception
Examples	`IOException`, `SQLException`	`NullPointerException`, `ArithmeticException`
Handling Required?	Yes (`try-catch` or `throws`)	No

Example:

```
// Checked Exception
void readFile() throws IOException {
FileReader file = new FileReader("file.txt");
}

// Unchecked Exception
void divide() {
int result = 10 / 0; // Throws ArithmeticException
}
```

Collections Framework

Q10: Difference between `ArrayList` and `LinkedList`?

Feature	ArrayList	LinkedList
Implementation	Dynamic array	Doubly linked list
Access speed	Fast (O(1) random access)	Slow (O(n) sequential search)
Insertion/Deletion	Slow (O(n) shifting required)	Fast (O(1) if at start/end)

Q11: Difference between `HashMap` and `TreeMap`?

Feature	HashMap	TreeMap
Ordering	No order	Sorted by keys
Speed	Faster (O(1) lookup)	Slower (O(log n))
Null Keys?	Allowed	Not allowed

Example:

```
HashMap<Integer, String> map = new HashMap<>();
map.put(1, "One");  // Unordered

TreeMap<Integer, String> treeMap = new TreeMap<>();
treeMap.put(1, "One");  // Sorted
```

---

5. Multithreading

Q12: How do you create a thread in Java?

Using `Thread` class:

```
class MyThread extends Thread {
```

```
    public void run() { System.out.println("Thread running..."); }
}

public class Main {
    public static void main(String[] args) {
        MyThread t = new MyThread();
        t.start();  // Starts new thread
    }
}
```

Using `Runnable` interface:

```
class MyRunnable implements Runnable {
    public void run() { System.out.println("Runnable thread
running..."); }
}

public class Main {
    public static void main(String[] args) {
        Thread t = new Thread(new MyRunnable());
        t.start();
    }
}
```
