Java Basics

1. What is java? Explain its features.

Ans. Java: java is a general-purpose computer programming language which is used to develop programs and web application. Java is an open source, platform independent and object oriented.

Features of java:

1) Simple
- Java is easy because it contains many features of C and C++ language.
- It doesn't include complex problems like pointer, goto statement and multiple inheritance.

2) Secure
- Java is known for security. It enables developers to develop virus free system.
- It provides features like secure class loading and bytecode verification for security.

3) Robust
- Java is known for reliability. It includes
  - Memory management
  - Garbage collection
  - Exception handling

4) Platform independent
- "Java source code compiled into bytecode which can run on any platform which have JVM installed" is knows as platform independent.

5) Object oriented
- Java follows object-oriented concepts like encapsulation, abstraction, polymorphism and inheritance.
- Java structure is a collection of classes and objects.

6) Multithreaded
- Thread is like separate programs which can run concurrently.
- Multithreaded makes easier to develop program which can perform multiple tasks.

7) Architecture neutral
- Java is architecture neutral because there is no implementation dependent on features.
- i.e.,

Language	**For 32-bit OS**	**For 64-bit OS**
C	Int occupy 2 byte	Int occupy 4 byte
Java	Int occupy 4 byte for both	

8) Interpreted
- Java bytecode is interpreted by JVM which converts it into machine level language which is suitable for operating system.

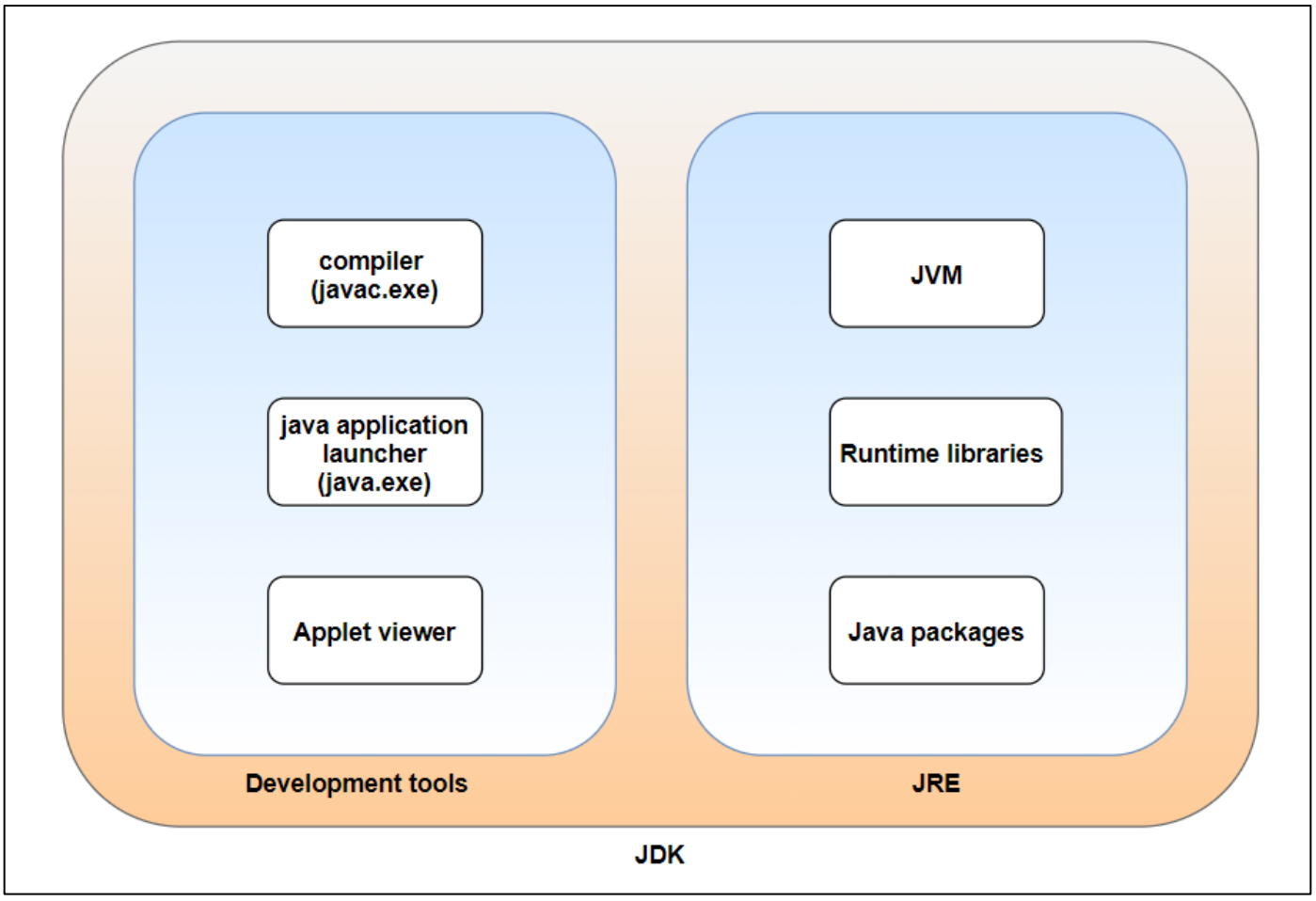
- Interpreted enables platform independent.

9) High performance
- Java has high performance because it uses JIT (just in time) compiler which compile bytecode into native code at runtime.

10) Distributed
- Distributed means created programs can run on multiple computers in network.
- Java provides tools and libraries like remote method invocation to help developer to built program that can communicate and share data across different machines.

2. Explain components of java.

Ans.



❖ **Java development kit**
- JDK consists of development tools and java runtime environment.
- JDK is software which is required to develop and execute java programs.
- It provides development tools like applet viewer, application launcher and compiler.

Java Basics

❖ **Java runtime environment**

- JRE includes java packages, runtime libraries and JVM.
- JRE doesn't contain any development tools like compiler and debugger.

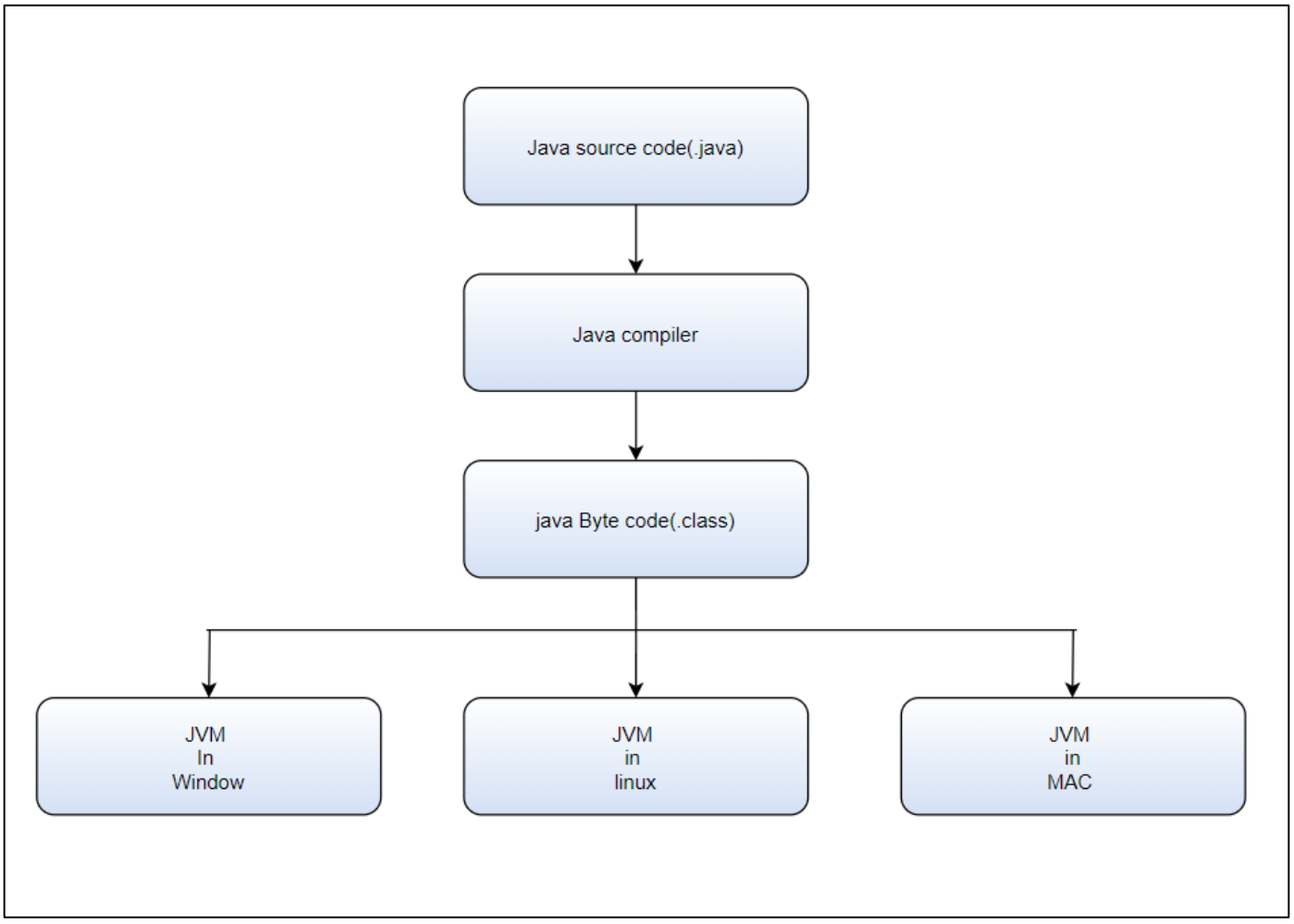
- It is a part of JDK but can be download separately.

❖ **Java virtual machine**

- JVM is used to convert bytecode into machine code.
- JVM provides interface which is not dependent on operating system.

❖ **Byte code**

- Byte code is an intermediate representation of source code.
- It is produced by compiler by compiling source code.
- The extension of byte code is ".class".



3. Variables, literals and datatypes.

Ans.

Variable: it is a storage unit which can hold the value of various data types.

Java Basics

❖ Rules for naming variable:
- It should start with lowercase.
- Avoid single character name like x, y, z.
- If name contains two words so it can display like empName.
- variable name shouldn't start with underscore or special character.

Literals: literals are value which is assign to a variable. It is a fixed value representation in code.

Types of literals:
1) integer
2) floating point
3) character
4) string
5) Boolean
6) null

Example: literals.java

Primitive Datatypes:
1) byte
- it is a smallest storage unit.
- Size: 8 bits
- Range: -128 to 127

2) Short
- It is least used data type.
- Size: 16 bits
- Range: -32768 to 32767

3) Int
- It is used to store non floating numbers.
- Size: 32 bits
- Range: -2,14,74,83,648 to 2,14,74,83,647

4) Long
- It is used to store large integer values.
- Size: 64 bits

5) char
- it is used to store character.
- Size: 16 bits

6) Float
- It is used to store single precious value.
- Size: 32 bits

7) Double
- It is used to store large floating number.
- Size: 64 bits

8) Boolean
- Boolean has two possible values: true or false
- Size of Boolean has not defined yet.

Example: datatypes.java

Nonprimitive data types:
1) array
2) string
3) class
4) interface

Frequently asked questions

1) Do strings in java require a null character at the end of the string like in C/C++?
- No, strings in Java do not require any null character at the end of the string.

2) Why is the char 2 byte in size in java?
- Other languages, such as C/C++, utilize just ASCII characters, and 8 bits is sufficient to represent all ASCII characters. However, Java uses the Unicode system, not the ASCII coding system, and 8 bits is insufficient to express all Unicode characters; hence Java requires 2 bytes for characters.

3) **What are the differences between static and instance variables?**
- All objects of a class have a single copy of static variables but a different copy of instance variables.
- Changes made to a static variable is reflected in all objects, whereas the changes made to an instance variable in an object has no effect on the other objects.
- Static variables can be accessed using the class name, and instance variables can be accessed through object references.

4. Operators in java.
Ans.
1) Arithmetic operators
2) Relational operators
3) Assignment operators
4) Logical operators

5) Increment and decrement
6) Conditional operators
7) Bitwise operators

❖ **Operator Precedence**

level	Operator	Category
1	() [] .	Postfix
2	++ -- ! ~	Unary
3	* / %	Multiplicative
4	+ -	additive
5	>> >>> <<	shift
6	> >= < <=	Relational
7	== !=	Equality
8	&	Bitwise and
9	^	Bitwise or
10	|	Bitwise OR
11	&&	Logical and
12	||	Logical or
13	?:	Conditional
14	= += -+ /+ %= >>= <<=	Assignment
15	,	Comma

Example: operators.java

Frequently asked questions

1) What is the function of operators?

- An operator is a tool that is used to change individual data elements and deliver a result. These things are referred to as operands or arguments. Special characters or keywords are used to represent operators.

2) How many types of unary operators are there?

- There are five types of unary operators. They are plus operator, minus operator, increment operator, decrement operator, and logical complement operator.

3) What is the difference between pre-increment and post-increment?

- In pre-increment operation, the value of the operand is first incremented by one and then used for any other purpose, but in post-increment, first the value of the operand is used and then incremented by one.

5. Control and loop statements

- **If statement:** The if statement is used to execute a block of code if a specified condition is true. It is the simplest form of control statement, allowing conditional execution based on a Boolean expression.

- **If else statement:** The if-else statement allows you to execute one block of code if the condition is true and another block of code if the condition is false.

- **If else if statement:** The if-else-if ladder allows you to test multiple conditions in sequence. As soon as one of the conditions is true, the corresponding block of code is executed, and the rest are skipped.

- **Nested if statement:** Nested if statements allow you to use an if or else block inside another if or else block. This is useful for checking multiple conditions in a hierarchical manner.

- **Switch case:** The switch statement in Java is a control flow statement that allows you to execute one block of code out of many possible blocks based on the value of a variable or expression. It provides a more readable and organized way to handle multiple conditions, especially when dealing with a series of potential values for a single expression.

- **For loop:** The for loop is used for iterating over a block of code a specific number of times. It is useful when you know how many times you want to execute a statement or a block of statements.

- **While loop:** The while loop executes a block of code as long as a specified condition is true. It's useful when you want to repeat a block of code but don't know beforehand how many times you need to repeat it.

- **Do while loop:** The do-while loop is similar to the while loop, but it executes the block of code at least once before checking the condition. The condition is evaluated after the execution of the loop body.

- **Break:** The break statement is used to exit a loop or switch statement immediately, regardless of the iteration or condition.

- **Continue:** The continue statement is used to skip the current iteration of a loop and proceed to the next iteration.

- **Nested loop:** Nested loops allow you to place one loop inside another loop. This is useful for working with multi-dimensional data structures or for performing repeated iterations at multiple levels.

Example: controlstatement.java

6. Explain type casting.

Ans. Type casting is a process which is used to assign the value of one datatype to variable of another data type.

There are two types of type casting:

1) Implicit typecasting(widening)

- Implicit type casting is performed by java itself. There is no require explicit command from programmer.
- There is no data loss.
- Following Conditions must satisfy to perform implicit typecasting
  → Both data types are compatible.
  → Destination data type is larger than source data type.

2) Explicit type casting(narrowing)
   - Explicit type casting is performed by programmers when they require.
   - There is data loss if source data type is larger than destination data type.

Example: typecasting.java

Frequently asked questions

1) What happens if you try to cast an incompatible type in Java?

- If you try to cast an incompatible type, Java throws a ClassCastException at runtime. This occurs when the object being cast is not an instance of the target class.

2) How does Java handle typecasting between objects? What is upcasting downcasting?

- Use upcasting (automatic) and downcasting (explicit) between classes in an inheritance hierarchy.
- **Upcasting:** Converting a subclass reference to a superclass reference. It's implicit and safe because the subclass is a type of superclass.
- **Downcasting:** Converting a superclass reference to a subclass reference. It's explicit and can lead to ClassCastException if not handled carefully.

Example

```
Animal animal = new Dog(); // Upcasting

Dog dog = (Dog) animal; // Downcasting
```

3) Scenario: You have a list of Object references that contain different types of data. How would you safely extract integers from the list?

- Use instanceof to check if an object is an instance of Integer before casting.

Example:

```
List<Object> list = Arrays.asList(1, "hello", 3.14, 42);

for (Object obj : list) {
```

```
    if (obj instanceof Integer) {
        Integer number = (Integer) obj;
        System.out.println("Integer found: " + number);
    }
}
```

7. Variable scope.

Ans. There are main four types of variable scope.

1) Local variable
- Variable can be declared within body of method known as local variable.

2) Instance variable
- Variable can be declared in class but outside the method known as instance variable

3) Static variable
- If a variable declared as static, a copy of that variable can't be created when object is created and it can access by its class name.

4) Block variable
- If a variable declared in any block like for loop, then it is known as block variable.

Example: scopeofvariable.java

Frequently asked questions

1) Practical question: give output of given code

```
public class ScopeTest {
    static int x = 10;
    int y = 20;

    public static void main(String[] args) {
        int x = 30;
        System.out.println("x: " + x); // Local x
        System.out.println("Static x: " + ScopeTest.x); // Static x

        ScopeTest obj = new ScopeTest();
        obj.printValues();
    }
```

```
  public void printValues() {
    int y = 40;
    System.out.println("Instance y: " + this.y); // Instance y
    System.out.println("Local y: " + y); // Local y
  }
}
```

Output

```
x:30
Static x:10
Instance y:20
Local y:40
```

8. Explain array.

Ans: array is a fixed size collection of elements of same data type.

- There are two types of arrays:

1) Single dimension array
2) Multi dimension array

Example: array.java

Array.java contains

- implementation of single dimension array
- implementation of multi dimension array
- some operation like merge, copy, reverse.

Frequently asked question:

1) What is the difference between an array and an ArrayList?

- Array: it is fixed size collection and can't store element of different data types.
- Arraylist: it is dynamic size collection and can store element of different data types.

2) Practical Question: How do you find the maximum product of two integers in an array?
File: PQ1.java

9. Explain methods with types.

Ans:

1) Predefined methods: are built in methods provided by java.
2) User defined methods: are created by programmer to perform specific task.
3) Instance methods: object of class is required to access instance method.
4) Static methods: we can access without using object of class.
5) Abstract methods: Abstract methods are declared without an implementation in an abstract class.

6) Final methods: final methods can't inherit by subclass

Types of methods by structure:

1) without return type and without parameters
2) without return type and with parameters
3) with return type and without parameters
4) with return type and with parameters

Frequently asked questions:

1) Can We Have a Method Inside a Method in Java?

- No, Java does not allow methods to be defined inside other methods. Methods in Java must be defined within a class, but not within another method. This is because Java does not support nested methods.

10. Explain classes and objects.

Ans:

- **Class**: A **class** is a **blueprint**, **template**, or **design** used to create **objects**.
  It defines:
  Properties (also called **fields** or **variables**)
  Behaviours (also called **methods** or **functions**)

- **Object**: An **object** is a **real instance** of a class. It is created based on the blueprint provided by the class. Each object has its own values for the properties.

Example: objectclass.java

Frequently asked questions:

1) Practical question: pass object of class as argument
File: PQ2.java

2) What happens when you assign one object reference to another?

- When you assign one object reference to another, both references point to the same object in memory. Any changes made through one reference will be reflected when accessed through the other.

11. Access modifiers.

Ans:

1) Public: public members can be accessed by other code from anywhere.
2) Private: private members can be accessed withing its class only.
3) Protected: it can be accessed withing same package and by subclass.
4) Default: if access modifier is not specified then it is work as "default". Can be accessed within package.

Example: accessmodifiers.java

Frequently asked questions:

1) Can a class be declared as private or protected?

- A top-level class cannot be declared as private or protected. However, nested (inner) classes can be declared as private, protected, or public.

2) What happens if you try to access a protected member from a class in a different package?

- A protected member can be accessed from a subclass in a different package, but it cannot be accessed from a non-subclass in a different package.

3) Can we declare constructors as private? If yes, why would we do that?

- Yes, constructors can be declared as private.

12. Explain getters and setters.

Ans:

- Getter: it is a method which is used to retrieve data from field.
- Setter: it is a method which is used to modifier data of field.

Example: gettersetter.java

13. Explain constructor with types.

Ans. Constructor is a one type of method which is used to initialize variables when object is created.

Characteristics:

1) it hasn't return type.
2) it is invoked when object is created.
3) constructors have same name as class name.
4) it can't be static, abstract, and final.
5) it can be overloading.

Example: constructors.java

Frequently asked questions

1) **What is the difference between a default constructor and a no-argument constructor?**

- Default constructor is created automatically if no constructor is defined in class. In default constructor, member has null value.
- If programmer creates a constructor which hasn't any argument is known as no-argument constructor. In this, member has not null value.

14. This keyword(shadow).

Ans. This keyword is used to refer the instance of current class.

Example: thiskeyword.java

15. Static keyword. (method hiding)

Ans. We can use static keyword with variable, method and as block.

Static variable:

- it can access by its class name and without creating an object of class.

Static method:

- it can access by its class name and without creating an object of class.
- Restrictions:
    - Static method must have static variable.
    - Super and this keyword can't use with static method.
    - Can call other static method only.

Static block:

- Static block initializes static variables. It is invoked when class in loaded into memory.

Example: statickeyword.java

Frequently asked questions:

1) **Can we override and overload static method?**

- We can overload static method because overload is based on compiler-time polymorphism and static methods are resolved at compile time.
- We can't override static method because override is based on runtime polymorphism and static methods are resolved at compile time.

2) **Can a class be declared as static?**

- we can't declare main class as static.
- We can declare inner class as static.

16. Mutable and immutable objects.

Ans:

- Mutable: if a value of an object can change then it is known as mutable object.
- Immutable: if a value of an object can't change then it is known as immutable object.

Example: mutableimmutable.java

17. Garbage collection

Ans: In java, garbage means unreferenced object.

- Garbage collection is a process of destroying unreferenced or unusual object.

- It is performed automatically.

- **How objects are considered as unreferenced?**

1) By assigning a null reference:
   example e1 = new example();
   e1=null;

2) By assigning a reference to another:
   example e2 = new example();
   example e3 = new example();
   e2 = e3;

3) Anonymous initialization:
   new example();

- **GC() method:**
  - it is used to perform garbage collection explicitly.
- **Finalize() method:**
  - If an object holds some no java resources like file handler, then you must make sure these resources are freed before object is destroyed.
  - Finalize method is invoked each time before the object is garbage collected.

Frequently asked questions:

1) What is garbage collection in Java, and why is it important?

- It is important because it prevents storage leaks and support efficient storage utilization.

18. Wrapper class.

Ans. We cannot use data types as objects in java.

- To overcome this java provides wrapper class, using wrapper class we use it as object.

Data types	Wrapper classes
byte	Byte
short	Short
int	Integer
long	Long
char	Character
float	Float
double	Double
boolean	Boolean

- Autoboxing: it is a conversion from primitive data type to object.

- Unboxing: it is a conversion from object to primitive data type.
- Important methods:
  → parseiInt(): it converts string to integer.
  → valueOf(): it is used to assign the value to the object of wrapper class.
  → intvalue(): it is used to access the value of objects of wrapper class.

Example: wrapperclass.java

Frequently asked questions:

1) How do wrapper classes improve code flexibility?

- Wrapper classes allow primitives to be treated as objects, which enables them to be stored in collections, passed to methods expecting objects, and allows for the use of utility methods provided by wrapper classes.

2) Can a wrapper class be null in Java?

- Yes, unlike primitives, wrapper class objects can be null, which can be useful in certain scenarios, such as when dealing with databases where a value might be absent.

19. Inheritance with types.

Ans. A class derives properties of another class known as inheritance.

- Parent class known as super class.
- Child class knows as sub class.

- Types of inheritance:
  1) Single inheritance: one child class inherits properties of parent class.
  2) Multilevel inheritance: assume there are three classes and C inherits properties of B, B inherits properties of A, so we can say that class c inherits properties of A.
  3) Hierarchical inheritance: here two class inherits properties of another class.

Example: mlinheritance.java (multilevel inheritance)
: hinheritance.java (hierarchical inheritance)

Frequently asked questions:

1) Explain constructor chaining.

- Constructor chaining is a process in which one constructor can call another constructor.
- This can be done in two ways:
  - In same class: it can be done using this keyword.
    Example: constructorchainingsc.java
  - In different classes (super-sub class): it can be done using super keyword.
    Example: constructorchainingdc.java

20. Super keyword.

Ans: super keyword can various task like

- accessing variable of super class
- accessing method of super class
- call constructor of super class

Example: superkeyword.java

Frequently asked questions:

1) **Can the super() call be used anywhere in the constructor?**

- No, the super() call must be the first statement in the constructor. If you do not explicitly call super(), the Java compiler automatically inserts a call to the no-argument constructor of the superclass.

2) **Can we use super to access a private member of the superclass?**

- No, super cannot be used to access private members of the superclass directly. Private members are not visible to subclasses.

21. Explain upcasting and down casting

Ans.

- **Upcasting:** Converting a subclass reference to a superclass reference. It's implicit and safe because the subclass is a type of superclass.
- **Downcasting:** Converting a superclass reference to a subclass reference. It's explicit and can lead to ClassCastException if not handled carefully.

Example: upcastingdowncasting.java

22. Explain instanceof operator.

Ans: it is used to check particular object is an instance of specific class/subclass of that class/interface/ or not.

Example: instanceofop.java

Frequently asked questions:

1) **Can you provide a scenario where instanceof is particularly useful?**

- instanceof is particularly useful when downcasting from a superclass to a subclass. It ensures that the object is of the correct type before performing the cast, thus avoiding ClassCastException.

23. Polymorphism with dynamic method dispatch

- Ans. **Polymorphism** means "**many forms**" — in Java, it refers to the ability of an object to **take many forms** or **perform different actions using the same name**.

- It has two types:

1) Method overloading
    - It is also known as compile time polymorphism.
    - When a same class has multiple method with same name and different signature so it is known as method overloading.
    - **Example:** methodoverloading.java
2) Method overriding
    - It is also known as runtime polymorphism.
    - When subclass and superclass both classes have same method with same name and same signature then subclass method overrides superclass method.
    - **Example:** methodoverloading.java

Dynamic method dispatch

- Dynamic method dispatch is a mechanism in which we can call overridden method.
- It is done by reference of super class.

Example: dynamicmethoddispatch.java

24. Final keyword

Ans. We can use final keyword with variable, method and class.

- If a variable is declared as final then it makes value of variable constant.
- If a method is declared as final then it can't be override by sub class.
- If a class is declared as final then it can't extend by another class.

Example: finalkeyword.java

Frequently asked questions:

1) **Can you declare a blank final variable and initialize it later? Show an example.**

Expected Solution:

```
class Example {
   final int MAX_VALUE;
   Example() {
      MAX_VALUE = 100; // Initialized in the constructor
   }
}
```

Explanation: A blank final variable (not initialized at the point of declaration) can be initialized in the constructor or an instance initialization block.

25. Abstraction with abstract class

Ans: abstraction is one of the fundamental concepts in java which is used to hide complexities.

- Abstract class can have both types of method abstract methods and normal methods.
- If you want to declare method only then you must declare method as abstract.
- If class has abstract method, then class must be declared as abstract.
- You can't create an object of abstract class.
- sub class must define abstract methods of super class.

Example: abstraction.java

Frequently asked questions.

1) **why is it important? How do you achieve it?**

- It is important because it simplifies the system complexities.
- We can achieve it by implementing abstract class and interface.

2) **Can abstract class have a constructor?**

- Yes, abstract class have constructor, it is called when object of sub class is created.

3) **What is concrete class?**

- A class which implements all methods of abstract class knows as concrete class

4) **Practical question: given an abstract class employee with an abstract method calculatesalary(), create subclasses fulltimeemployee and parttimeemployee that implement this method.**

- **File name:** PQ3.java

26. Explain interface with multiple inheritance.

Ans. Interface is a same as abstract class.

- but interface contains all abstract methods.
- we can implement multiple inheritance using interface.
- An interface can extend another interface

Example: Design an interface Payment with an abstract method pay(). Implement this interface in two classes, CreditCardPayment and PayPalPayment, each providing a specific implementation of the pay() method.

File name: interfaceexample.java

Example of multiple inheritance using interface: minheritance.java

27. Explain package.

Ans. A package is a namespace for organizing classes and interfaces in a logical manner.

- Packages are used to avoid name conflicts, control access, and maintain a structured organization of code.
- Java package is used to create directory to store java files and used to retrieve that directory.
- We can perform two operations on java packages:
  1) Create package
     Example: com/example/createpackage.java
  2) Import package
     Example: importpackage.java

Frequently asked questions:

1) Explain the access control provided by packages.

- Classes, methods, and variables declared without any access modifier have package-private access, meaning they can only be accessed by other classes within the same package.

28. Enum class

Ans. an Enum (short for enumeration) is a special data type that represents a fixed set of constants.

- An Enum class in Java is used to define a collection of constants that belong together.
- It is a type-safe way to define and use named constants, making your code more readable and maintainable.

- **Example:** Enumexample.java

29. Inner classes.

Ans. A class which is defined within another class known as inner class.

There are 4 types of inner class.

1) Member inner class
- a class which is defined within class but outside the method is known as member inner class.
- **Example:** memberinnerclass.java

2) Static nested class
- A class which is defined within another class and declared as static, it is known as static nested class.
- It can access the static member of outer class directly but creating object is required to access the non-static members.
- **Example:** staticnestedclass.java

3) Local inner class
- A class which is defined within class and within method known as local inner class.
- **Example:** localinnerclass.java

4) Anonymous inner class

- An Anonymous Inner Class is a class that is defined and instantiated in a single expression. It is often used to provide a quick implementation of an interface or a subclass.
- **Example:** anonymousinnerclassexample.java

Frequently asked question:

1) Why would we use inner class?

- Inner classes are used to grouping classes to that will be used to provide better encapsulation.

2) Why would we use anonymous inner class?

- anonymous inner classes are used when we need one time implementation.

30. Biginteger and bigdecimal.

Ans. Biginteger and bigdecimal are two classes which are provided by "java.math" package which are used to perform mathematical operation between large values.

Example: bigintegerbigdecimal.java

❖ **Scale and rounding mode:**

- Scale: it represents the number of digits at right side of floating point.
- Rounding mode: there are three rounding mode 1) RoundingMode.HALF_UP
  2) RoundingMode.HALF_DOWN
  3) RoundingMode.HALF_EVEN
- **Example:** scaleandrouding.java

31. String class, Stringbuffer class, Stringbuilder class with constructors and methods

String: stringclass.java
String buffer: stringbufferclass.java
String builder: stringbuilderclass.java

Frequently asked questions:

1) How does the intern() method work in String?

```
public class StringInternExample {
   public static void main(String[] args) {
      String str1 = new String("Hello");
      String str2 = "Hello";
      String str3 = str1.intern();

      System.out.println(str1 == str2);  // false
      System.out.println(str2 == str3);  // true
```

```
  }
}
```

- Str1 object is created using new keyword, so it is a new object in memory.
- Str2 is string literal, it is stored in string pool.
- (str1 == str2) is equal to false because both are stored in different location.
- (str2 == str3) is equal to true because str1 is transferred into string pool using intern method and it is referred as str3.

2) Difference between string, string buffer, string builder

String class	String buffer class	String builder class
Immutable	Mutable	Mutable
Stored in heap memory or string constant pool	Stored in heap memory	Stored in heap memory
Synchronization: No	Yes	No

3) Practical question: reverse the string without using inbuilt function

- **File name:** PQ4.java

4) Practical question: check given string is palindrome or not using string buffer.

- **File name:** PQ5.java

5) Conversion from (String, Stringbuffer, Stringbuilder) to (string, Stringbuffer, Stringbuilder)

- **File name:** stringconversion.java

32. What is exception handling? with types.

Ans. exception handling is a mechanism which is used to handle runtime errors to maintain execution flow.

❖ **Types of exception:**

1) Checked exception

- Exception which is checked at compile time known as checked exception.
- IOException
- SQLException
- FileNotFoundException
- ClassNotFoundException
- ParseException

2) Unchecked exception

- Exception which is checked at runtime known as unchecked exception.
- ArrayIndexOutOfBoundsException
- NullPointerexception

- ArihmaticException
- IllegalArgumentException
- NumberFormatException

❖ **Structure:**

```
try{
        // block in which exception can occur.
}
catch{
        // block in which exception can handle.
}
finally{
        // it will execute always, exception are occurred or not it doesn't matter
}
```

Example: exception.java

33. Final / finally / finalize

final	**finally,**	**finalize**
it is a keyword	It is a block	it is a method provided by object class.
Used to make variable constant, prevent method to override And prevent class to inherit	Finally block will execute always, exceptions are occurred or not doesn't matter. This block is used to execute important code like memory cleanup.	If an object holds some no java resources like file handler, then you must make sure these resources are freed before object is destroyed.

Ex:
Final int var =3.14;
Final void method () { }
Final class example{ }
	Ex:
	Finally {
	// code
	}
		Ex:
		Protected void finalize () {
		//code
		}

34. Throw and throws keyword.

Ans. throw keyword is used to throw exception explicitly from method. throw keyword can throw checked and unchecked exception.

- throws keyword is used in method signature to declare method can throw one or more exception.

Example: throwandthrows.java

35. How to make custom exception.

Ans. there are two steps two create custom exception.

1) Create a class and Extend the Exception class
2) Create constructor of subclass

1) File name: customException.java

36. Methods of exception class.

Ans.

1) getMessage(): Retrieves the exception message.
2) toString(): Returns a string representation of the exception.
3) printStacktrace(): Prints the stack trace to the console.
4) getStackTrace(): Retrieves the stack trace elements and prints them.

2) Example: customException.java

37. File class with methods and constructor.

Ans. file class is provided by **java.io** package and it is used to create, delete and manipulate files and directories on file system.

❖ **Constructors:**

1) File file = new File(String path)
2) File file = new File(String parent, String child)
3) File file = new File(File parent, String child)

❖ **Methods:**

1) file.exists()
2) file.createNewFile()
3) file.mkdir()
4) file.mkdirs()
5) file.getName()
6) file.isFile()
7) file.isDirectory()
8) file.length()
9) file.renameTo(String Newpath)
10) file.getAbsolutePath()
11) file.getPath()
12) file.getParent()
13) file.canRead()
14) file.canWrite()
15) file.canExecute()
16) file.delete()
17) file.list(): returns in array (String[])
18) file.listFiles(): returns in array (File[])

Example: fileexample.java

Frequently asked question:

1) **Practical question: program to search file in directory and subdirectory.**
   - **File name:** PQ6.java

38. Explain byte stream.

Ans. byte stream is used to perform input output of 8-bit bytes. All classes are derived from inputStream and outputStream.

1) **Input stream**
   - It is used to read data from source.
   - Methods:
     - → int read (): reads one byte of data and returns -1 at the end of the stream.
     - → int read (byte[] b): reads byte into array and returns the number of bytes read.
     - → int available (): it returns the available bytes.
     - → void close (): close the input stream.
   - Subclasses:
     - → FileInputStream: reads bytes from file.
     - → BufferedInputStream: add buffering in input stream for efficiency.
     - → ByteArrayInputStream: reads bytes from array.
     - → DataInputStream: reads primitive data types from input stream.

2) **Output stream**
   - It is used to write data to source.
   - Methods:
     - → void write (int c): writes one byte of data.
     - → void write (byte[] b): writes an array of bites.
     - → void flush (): forces any output byte to be written.
     - → void close (): close the output stream.
   - Subclasses:
     - → FileOutputStream: writes bytes to a file.
     - → BufferedOutputStream: add buffering in output stream for efficiency.
     - → ByteArrayOutputStream: writes bytes to array
     - → DataOutputStream: writes primitive data types to output stream

❖ **FileInputStream and FileOutputStream:**

- **FileInputStream:**

  Constructors:
  - → FileInputStream(File file)
  - → FileInputStream(String name)

  Methods:
  - → int read()
  - → int read(byte[] b)

→ int read(byte[] b, int off, int length)
→ int available()
→ skip(long n)
→ void close()

- **FileOutputStream:**
  Constructors:
  → FileOutputStream(File file)
  → FileOutputStream(String name)
  → FileOutputStream(File file, boolean append)
  → FileOutputStream(String name, boolean append)

  Methods:
  → void write(int c)
  → void write(byte[] b)
  → void write(byte[] b, int off, int length)
  → void append(String string)
  → void flush()
  → void close()

- **Example:** fileiostream.java

❖ **BufferedInputStream and BufferedOutputStream:**

- **BufferedInputStream**
  Constructors:
  → BufferedInputStream(FileInputStream fis)
  → BufferedInputStream(FileInputStream fis, int size)

  Methods:
  → int read()
  → int read(byte[] b, int off, int length)
  → int available()
  → void close()
  → void mark(int n)
  → void reset()

- **BufferedOutputStream**
  Constructors:
  → BufferedOutputStream(BufferedOutputStream fos)
  → BufferedOutputStream (BufferedOutputStream fos, int size)

  Methods:
  → void write(int c)

→ void write(byte[] b, int off, int length)
→ void flush()
→ void close()

- **Example:** bufferediostream.java

❖ **ByteArrayInputStream and ByteArrayOutputStream:**

- **ByteArrayInputStream**

Constructors:

→ ByteArrayInputSream(byte[] b)

Methods:

→ int read()
→ int read(byte[] b, int off, int length)
→ int available()
→ void reset()
→ void skip(int n)

- **ByteArrayOutputStream**

Constructors:

→ ByteArrayOutputStream()

Methods:

→ void write(int c)
→ void write(byte[] b, int off, int length)
→ void writeTo()
→ byte[] toByteArray()
→ void reset()
→ int size()

- **File name:** bytearrayiostream.java

39. Explain serialization and De-serialization.

Ans.

Serialization:

- Serialization is the process of converting a Java object into a byte stream, making it easy to save the object's state to a file, send it over a network, or store it in a database. In Java, an object is serialized by implementing the Serializable interface.
- To enable serialization in java:
  - ➔ The class must implement Serializable interface
  - ➔ The Serializable interface contains no methods
  - ➔ ObjectOutputStream and ObjectInputStream are used for serialization and Desirialization.

Deserialization:

- Deserialization is the reverse process of serialization. It takes the byte stream generated during serialization and converts it back into a Java object, reconstructing the object with the same state it had during serialization.

Example: Serialization.java

40. Character stream.

Ans. character stream is used to perform input and output operation on data of character. It fetched raw bytes to perform operation. All classes are derived from Reader and Writer.

❖ **FileReader and FileWriter:**

○ **FileReader:**

Constructor:

→ FileReader(File file)

→ FileReader(String name)

Methods:

→ int read()

→ int read(byte[] b , int off, int length)

→ void close()

○ **FIleWriter:**

Constructor:

→FileWriter(File file)

→ FileWriter(String name)

Methods:

→ void write(int c)

→ void write(char[] c , into off, int length)

→ void write(String str)

→ void flush()

→ void close()

○ **Example:** filereaderwriter.java

❖ **BufferedReader and BufferedWriter:**

○ **BufferedReader:**

Constructor:

→ BufferedReader(Reader rd)

→ BufferedReader(Reader rd, int size)

Methods:

→ int read()

→ int read(char c, int off, int length)
→ String readLine()
→ boolean ready()
→ void close()

- **BufferedWriter:**

Constructor:

→ BufferedWriter(Writer wr)
→ BufferedWriter(Writer wr, int size)

Methods:

→ void write(int c)
→ void write(char[] c, int off, int length)
→ void write(String s, int off, int length)
→ void newline()
→ void flush()
→ void close()

- **Example:** bufferedreaderwriter.java

❖ **inputStreamReader and OutputStreamWriter:**

- **inputStreamReader**

Constructor:

→ InputStreamReader(InputStream in)
→ InputStreamReader(InputStream in, String charsetname)
→ InputStreamReader(InputStream in, charset cs)

Methods:

→ int read()
→ int read(char[] c, int off, int length)
→ boolean ready()
→ void close()

- **InputStreamWriter**

Constructor:

→ OutputStreamWriter(OutputStream out)
→ OutputStreamWriter(OutputStream out, String charsetname)
→ OutputStreamWriter(OutputStream out, charset cs)

Methods:

→ void write(int c)
→ void write(char[] c, int off, int length)

→ void writer(String str, int off, int length)
→ void flush()
→ void close()

- **Example:** inputstreamreaderwriter.java

❖ **StringReader and StringWriter:**

- **StringReader**

Constructor:

→ StringReader(String s)

Methods:

→ int read()
→ int read(char[] c, int offset, int length)
→ long skip(int n)
→ boolean ready()
→ void mark()
→ void reset()
→ void close()

- **StringWriter**

Constructor:

→ StringWriter()
→ StringWriter(int size)

Methods:

→ void write(int c)
→ void write(char[] c, int off, int length)
→ void write(String s)

→ void write(String s, int off, int length)
→ append(char c)
→ append(charSequence cr)
→ getBuffer()
→ void flush()
→ void close()

- **Example:** stringreaderwriter.java

41. Collection framework

1) Arrays class

- Arrays class is provided by "java.util" package. It provides various methods to manipulate array.

- **Methods:**

→ asList(array)
→ sort(array)
→ binarySearch(array, key)
→ equals(array1, array2)
→ toString(array)
→ hashCode(array)
→ fill(array, value)
→ copyOf(original array, new length)
→ copyOfRange(original array, start, end)

- **Example:** arraysclass.java

2) Collection class

- Collections class is provided by "java.util" package. It provides various methods to manipulate list.

- **Methods:**
  → sort(list)
  → copy(list)
  → frequency(element)
  → min(list)
  → max(list)
  → replaceAll(list, old, new)
  → reverse(list)
  → unmodifiable(list)
  → disjoint(list1, list2)

- **Example:** collectionsclass.java

3) List

a) Arraylist

- It is a resizable array implementation of the List interface.
- Size of Arraylist is not fixed.
- Internally, default size of that array list is 10, when an array list is created. If all elements are filled to size, then size of Arraylist is incremented by (n + n/2 +1).

- **How to create object:**
  ArrayList<type> name = new ArrayList<>();

- **Methods:**
  → add(element)
  → add(index, element)
  → addAll(array list)
  → size()

→ contains(element)
→ remove(index)
→ remove(element)
→ get(index)
→ set(index, element)
→ isEmpty()
→ indexOf(element)
→ clear()

- **Example:** arraylistexample.java

b) Vector class

- Vector class is dynamic implementation of array.
- If your application requires thread-safe operations on a list, Vector is a suitable choice.

- **Constructors:**
  → List<Integer> lst = new ArrayList<>(Arrays.asList(10,30,70,60,40));
  Vector<Integer> vct = new Vector<>(lst);
  → Vector<>(); // with initial capacity 10
  → Vector<>(20);// with initial capacity 20
  → Vector<>(20,8); // with initial capacity 20, and capacity increment is 8

- **Example:** vectorclass.java

4) Set

a) Hashset

- Hashset is provided by collection framework.
- It is used to store collection of unique elements.

- **Constructor:**
  → HashSet<>()
  → HashSet<>(initial capacity)
  → HashSet<>(list)

- **Characteristics:**
  → no duplication
  → no order
  → allows null value

- **How to create:**
  HashSet<type> name = new HashSet<>();

Remove Duplicate Characters from a String Using StringBuilder

Program to Remove Duplicates from an ArrayList

b) Treeset

- Treeset is provided by collection framework.

- **Constructor:**
  → TreeSet<>()
  → TreeSet<>(initial capacity)
  → TreeSet<>(list)
  → TreeSet<>(comparator obj)
  → TreeSet<>(sortedSet)

- **Characteristics:**
  → no duplication
  → in sorted order
  → does not allow null value

- **Methods**
  → first()
  → last()
  → contains()
  → size()
  → boolean add(element)
  → boolean remove(element)
  → ceiling(num)
  → floor(num)
  → headSet(element)
  → tailSet(element)
  → descendingSet()
  → subset(element, element)

- **Example:** treesetexample.java

5) Map interfaces

a) Hashmap

- HashMap is a part of the Java Collections Framework and is found in the **java.util** package. It implements the Map interface and is used to store key-value pairs.

- **Constructors:**
  → HashMap<>()
  → HashMap<>(20); // with initial size:20
  → HashMap<>(anothermap);

- **Characteristics**
  → unordered
  → fast
  → Non-synchronized

- **Methods:**
  → put(key, value)
  → putIfAbsent(key, value)
  → get(key)
  → containsKey(key)
  → containsValue(value)
  → foreEach(k,v) -> SOP(k,v)
  → replace(key, value)
  → remove(key)
  → entrySet()
  → keyset()
  → values()
  → isempty()
  → size()
  → clear()

- **Example:** hashmapexample.java

b) Treemap

- TreeMap is a part of the Java Collections Framework and is found in the **java.util** package. It implements the Map interface and is used to store key-value pairs.

- **Constructors:**
  → TreeMap<>()
  → TreeMap<>(anothermap)
  → TreeMap<>(Comparator.naturalOrder)
  → TreeMap<>(Comparator.sortedOrder)

- **Characteristics**
  → ordered
  → slow
  → synchronized

- **Methods:**
  → put(key, value)
  → putIfAbsent(key, value)
  → get(key)
  → containsKey(key)

→ containsValue(value)
→ firstKey()
→ lastKey()
→ Map<Integer, String> map = map.HeadSet(to)
→ Map<Integer, String> map = map.TailSet(from)
→ Map<Integer, String> map = map.SubSet(from, to)
→ foreEach(k,v) -> SOP(k,v)
→ replace(key, value)
→ remove(key)
→ entrySet()
→ keyset()
→ values()
→ isempty()
→ size()
→ clear()

- **Example:** treemapexample.java

a. Iterator

- Java collection framework provides away to traverse the array list and hash map know as iterator.

- **How to create:**
  → Iterator<type> name = arraylist.iterator();

- **Methods**
  → hasNext()
  → next()
  → remove()

- **Example:** iteratorexample.java

b. **Comparator**

- The Comparator interface in Java is used to define the order of objects.

- We can use comparator in two ways:
  1) By implement comparator interface

```
class namecompincomparatorexample implements Comparator<studentincomparatorexample>{
    public int compare(studentincomparatorexample s1, studentincomparatorexample s2){
        return s1.getName().compareTo(s2.getName());
    }
}
```

```
Collections.sort(stlist, new namecompincomparatorexample());
        System.out.println("\nSorted list by name:");
        for(studentincomparatorexample element : stlist){
            System.out.println(element.getEnroll() +" -> "+ element.getName());
        }
```

2) By creating object

```
 Comparator<studentincomparatorexample> compareenroll = new Comparator<
studentincomparatorexample>() {
            public int compare(studentincomparatorexample s1, studentincomparatorexample s2){
                return Integer.compare(s1.getEnroll() , s2.getEnroll());
            }
        };

        Collections.sort(stlist, compareenroll);
        System.out.println("\nSorted list by enroll:");
        for(studentincomparatorexample element : stlist){
            System.out.println(element.getEnroll() +" -> "+ element.getName());
        }
```

42. Stream API

Ans. The **Stream API** in Java, introduced in **Java 8**, is a powerful tool for processing collections of data in a functional and declarative style. It allows you to manipulate and process data in a pipeline-like structure, making code cleaner and more readable.

- **Example:** Streamaipexample.java

43. Thread introduction, thread cycle, thread priorities

Ans. A **thread** in Java is a lightweight process that allows a program to perform multiple tasks simultaneously.

- It is provided by java.lang package.
- **We can create thread in two ways.**
    1) By extending Thread class.
    - **Example:** ThreadExample1.java
    2) By creating object of Thread class.
    - **Example:** ThreadExample1.java
    3) By implementing Runnable interface.
    - **Example:** ThreadExample2.java

- **Methods:**
  - start()
  - join()
  - isAlive ()
  - sleep(mili seconds)
  - getPriority()
  - setPriority(n) : n should be 1 to 10
  - currentThread()
  - getName()

- **Thread Cycle:**
  - New State
  - Runnable state
  - Running state
  - Waiting state
  - Terminated state

- **Thread Priority:**
  - In Java, thread priority is a mechanism to suggest the relative importance of a thread to the Java Virtual Machine (JVM) and the operating system's thread scheduler.
  - **Priority Levels:** Java threads have priorities that can range from Thread.MIN_PRIORITY (1) to Thread.MAX_PRIORITY (10). By default, threads are assigned the priority Thread.NORM_PRIORITY (5).
    → **Thread.MIN_PRIORITY**: The lowest priority (value 1).
    → **Thread.NORM_PRIORITY**: The default priority (value 5).
    →**Thread.MAX_PRIORITY**: The highest priority (value 10).
  - **Example:** ThreadPriorityExample.java

- **Synchronization**
  - When multiple threads access shared resources concurrently, it can lead to inconsistent data or race conditions. Java provides synchronization to avoid this.
  - **Example:** SynchronizationExample.java

- **Practical Question:** Perform synchronization on bank account.
  - **File:** PQ7.java

44. Deadlock

Ans. Deadlock is a situation in concurrent programming where two or more threads are blocked forever, each waiting for the other to release a lock. This often occurs in a situation where multiple threads need the same set of resources but obtain them in different order.

- **Example:** DeadlockExample.java
  It contains:
  → deadlock occurring code
  → deadlock avoiding code

45. Suspending, resuming and stopping thread

Ans.

- **Example:** Threadoperation.java

Detailed Table with Second-wise Explanation

Time (Seconds)	Thread Activity	Code Execution	Output
0	`tm1` thread is started	`tm1.start()` invokes `run()` method	"Thread 1 is running."
1	`tm1` is still running	Loop continues in `run()`	"Thread 1 is running."
2	`tm1` is still running	`Thread.sleep(2000);` delays next command	"Thread 1 is running."
2	Main thread calls `Suspend()`	`suspendflag` is set to `true`	"Thread 1 is suspended."
3	`tm1` is suspended	Waiting inside `synchronized(this)` block	No output (thread is paused)
4	Main thread calls `Resume()`	`suspendflag` is set to `false` and `notify()` is called	"Thread 1 is resumed."
5	`tm1` resumes running	Loop continues in `run()`	"Thread 1 is running."
6	Main thread calls `Stop()`	`stopflag` is set to `true`	"Thread 1 is stopped."
7	`tm1` thread stops	Exits the loop, prints "Thread 1 is existing."	"Thread 1 is existing."

46. Generics

Ans. Java generics is a feature that allows you to create classes, methods, and interfaces that can operate on different data types while providing compile-time type safety.

- Instead of writing the same code multiple times for different types, you write a single generic code that works with any type.
- **Example:** genricsexample.java
  it contains:
  → implementation of generics class
  → implementation of generics method
  → implementation of bounded generics
- Bounded generics is used to prevent datatypes in generics class and generics method.

Generics wildcards

1) ? (unbounded)

- An unbounded wildcard can accept any type. This is useful when you don't need to impose any restrictions on the type.
- Example:

```
public static void printList(List<?> list) {
   for (Object obj : list) {
      System.out.println(obj);
   }
}
```

2) ? extends (upper bounded)

- An upper bounded wildcard restricts the unknown type to be a specific type or a subclass of that type. It is used when you want to read from a structure but not modify it.
- Example**:**

```
public static void printNumbers(List<? extends Number> list) {
   for (Number num : list) {
      System.out.println(num);
   }
}
```

3) ? super (lower bounded)

- A lower bounded wildcard restricts the unknown type to be a specific type or a superclass of that type. It is used when you want to write into a structure.
- Example:

```
public static void addNumbers(List<? super Integer> list) {
   list.add(1);  // You can add an Integer to the list
   list.add(2);
}
```

Java DataBase Connectivity

1. what is JDBC

Ans. JDBC stands for Java Database Connectivity.

- JDBC is an API which allows you to interact with databases.
- It is a set of various classes, interfaces that enables you to connect and communicate with different databases like MySQL, Oracle etc.
- It allows java programs to
  - → Establish the connection
  - → Execute SQL queries
  - → Retrieve and manipulate data
  - → Manage transaction (Commit / rollback)

2. JDBC architecture and components

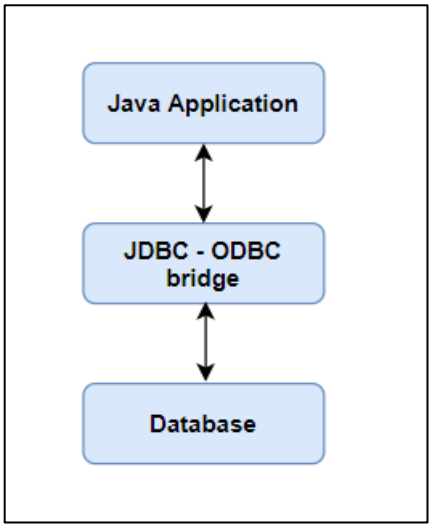
Ans. Key components are:

1) JDBC driver

- There are four types of JDBC driver.

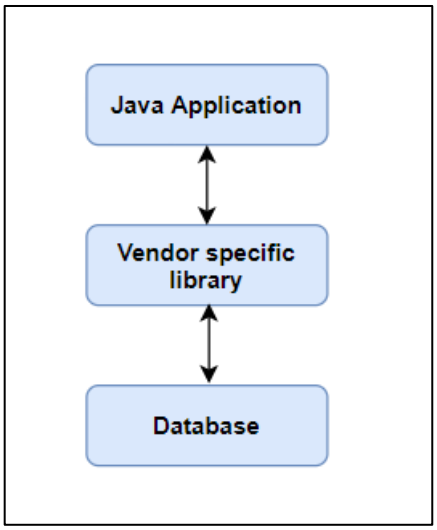
(i) Type 1: JDBC to ODBC driver

- It acts as bridge between the JDBC API and ODBC, allowing java application to connect to databases that supports ODBC.



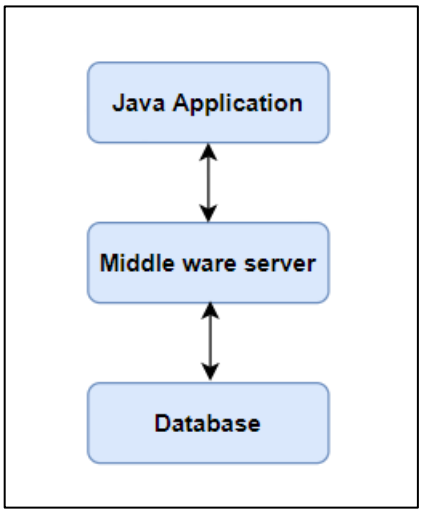
(ii) Type 2: Native – API driver (vendor specific)

- It uses the native libraries of database (vendor specific libraries) to connect to the database.
- It directly converts JDBC calls into API calls which the database understand.



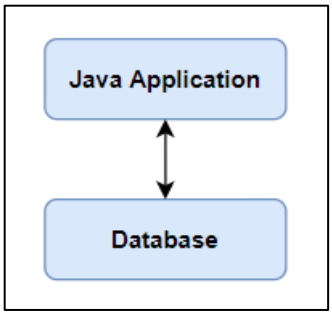
(iii) Type 3: Network protocol driver

- It uses three tier architecture to connect to database. It communicates with a middle ware application server (web logic) which interact with database.



(iv) Type 4: Thin driver (Direct to database)

- It directly converts JDBC calls into the database's native protocol.



2) JDBC driver manager

- JDBC driver manager is a class and part of API which is used to
  → Load driver

→ Establish the connection
→ Managing list of drivers and ensuring the correct driver is used.

3) Connection interface

- This represents the physical connection to a database.
- It is created using DriverManager.getConnection(URL, username, password);

4) Statement interface

- This interface allows you to send SQL queries to database.
- There are three types:
  → Statement: used for queries without parameters.
  → PreparedStatement: used for queries with parameters.
  → CallableStatement: used to execute stored procedure.

5) Resultset interface

- This represents the result set obtained from SQL queries.

6) Exception handling

- JDBC uses SQLException to handle database related errors.

3. Steps to use JDBC

Ans.

1) Load the JDBC driver
2) Establish the connection
3) Create a statement
4) Execute query
5) Process result
6) Close resources